UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAîTRISE EN ÉLECTRONIQUE INDUSTRIELLE

PAR
MARTIN DE MONTIGNY

GÉNÉRATION AUTOMATIQUE DE CODE POUR LA MODÉLISATION, SIMULATION ET COMMANDE DE ROBOTS EN TEMPS RÉEL

MARS 2000

RÉSUMÉ

Le temps consacré à la modélisation et à l'écriture du code de simulation constitue un problème majeur lors de la conception de systèmes robotiques. Nous avons développé un outil de modélisation et de simulation de système robotiques. À partir d'une brève description géométrique et dynamique d'un robot entrée dans une interface graphique, l'outil calcule, en effectuant des manipulations symboliques, les équations cinématiques et dynamiques du modèle du robot. Les modèles peuvent être produits sous la forme récursive de Newton-Euler ou sous la formulation en boucle fermée de Lagrange-Euler. Nous montrons les avantages et inconvénients des deux formes. Ensuite, les équations symboliques du modèle du robot sont incluses dans les fichiers de simulation à l'aide de manipulations de chaînes de caractères et de fichiers. Le générateur de code produit aussi les librairies de blocs d'appel des codes produits ainsi qu'un contrôleur adaptatif de position.

Pour valider le générateur de code, nous modélisons, commandons et simulons une main robotique saisissant un objet flexible. Cette étude de cas est représentative d'applications réelles en robotique. L'algorithme de commande adaptative de Slotine et Li est employé pour réaliser un contrôleur de position adaptatif à la variation des masses des doigts pendant les mouvements sans contraintes. Un contrôleur de force proportionnel intégral non linéaire modifie la trajectoire de position lors des contacts afin de rencontrer les objectifs de saisie. Nous avons développé un superviseur pour automatiser l'opération des contrôleurs, des générateurs de trajectoire et de l'identificateur. Nous avons

programmé une stratégie de saisie de façon à obtenir un bon contrôle de force (erreur minimale) tout en minimisant l'amplitude et la durée du régime transitoire de contact entre les doigts et l'objet. Nous séparons le code séquentiel obtenu pour le simuler sur un système multi-PC, ce qui nous permet de garder une bonne précision tout en simulant en temps réel. Les résultats de simulation montrent que les contrôleurs conçus sont stables et permettent d'obtenir des erreurs de position et de force minimes en présence d'erreur de modélisation cinématique et de variations de rigidité de l'objet saisi.

REMERCIEMENTS

J'aimerais particulièrement remercier mon directeur de recherche, le professeur Pierre Sicard, pour ses précieux conseils et son temps. Ce travail a été financièrement supporté par Opal-rt Technologies inc. et par la bourse Intervention Spéciale de l'UQTR. Je remercie Michel Lambert pour son aide à la simulation en parallèle de nos algorithmes. Je remercie Brian Moore pour son aide à la programmation de certains algorithmes.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SYMBOLES

a Accélération calculée par le générateur de trajectoire

$\mathbf{a}$ Vecteur des paramètres réels

$\hat{\mathbf{a}}$ Vecteur des paramètres estimés

$\dot{\hat{\mathbf{a}}}$ Vecteur de la dérivée des paramètres estimés

a_a Consigne d'accélération filtrée calculée par le contrôleur de force

a_{i-1} Distance de Z_{i-1} à Z_i mesurée le long de X_{i-1}

$\mathbf{C}$ Matrice des termes centrifuges et de Coriolis

d Amortissement de l'effecteur dans l'espace cartésien (modèle du système robot-environnement servant à la conception du contrôleur de force)

D Dérivée de l'accélération (utilisée par le générateur de trajectoire)

d_i Distance de X_{i-1} à X_i mesurée le long de Z_i

e Erreur de force

$\mathbf{e}$ Erreur de prédiction de l'identificateur

e_c Erreur cinématique

$e_{\max F}$ Erreur absolue maximale en force

$e_{\max Px}$ Erreur absolue maximale de position en x

$e_{\max Pz}$ Erreur absolue maximale de position en z

$e_{\max \theta}$ Erreur absolue maximale d'orientation de l'effecteur

e_{moyF} Erreur absolue moyenne en force

e_{moyPx} Erreur absolue moyenne de position en x

e_{moyPz} Erreur absolue moyenne de position en z

$e_{moy\theta}$ Erreur absolue moyenne d'orientation de l'effecteur

e_s Erreur de force signée

eff Efficacité de la simulation en parallèle

$F(s)$ Force externe s'opposant au mouvement de l'effecteur (modèle du système robot-environnement pour la conception du contrôleur de force)

F Force de contact entre le robot et l'environnement

F_{2x} Contribution en force appliquée sur l'objet du doigt 2 en x par rapport à la base de l'environnement

F_{3x} Contribution en force appliquée sur l'objet du doigt 3 en x par rapport à la base de l'environnement

F_{2y} Contribution en force appliquée sur l'objet du doigt 2 en y par rapport à la base de l'environnement

F_{3y} Contribution en force appliquée sur l'objet du doigt 3 en y par rapport à la base de l'environnement

${}^i\mathbf{f}_i$ Force exercée par le membre i-1 sur le membre i représenté dans le référentiel i

$\mathbf{F}_i$ Force d'inertie agissant sur le membre *i*

$\mathbf{F_e}$ Vecteur de forces externes agissant sur l'effecteur

F_r Force de consigne actuelle

F_{row} Force appliquée par le robot sur l'objet, dans la direction *w* (*x* ou *y*)

F_s Force de consigne qui était appliquée avant le régime transitoire permettant d'atteindre la nouvelle force de consigne actuelle F_r

F_v Frottement visqueux caractérisant le déplacement de l'objet

g Constante gravitationnelle

$G(s)$ Fonction de transfert constituant le modèle de deuxième ordre représentant le robot et son environnement (utilisé pour la conception du contrôleur de force)

$\overline{G}(s)$ Fonction de transfert du robot en boucle ouverte

$\mathbf{G}$ Vecteur des termes de gravité

${}^0\mathbf{g}$ Vecteur de gravité par rapport à la base du robot

$\mathbf{I}$ Matrice identité

${}^{Ci}\mathbf{I}_i$ Tenseur d'inertie au centre de masse du membre i représenté dans le référentiel i

$\mathbf{J}(\mathbf{q})$ Jacobien de vitesse

$\mathbf{J}(\mathbf{q})^{-1}$ Jacobien de vitesse inverse

$\mathbf{J}(\mathbf{q})^{\mathrm{T}}$ Jacobien de vitesse transposé (Jacobien de force)

$\mathbf{J}(\mathbf{q})^{*}$ Inverse généralisé du Jacobien de vitesse

$\mathbf{J}_F(\mathbf{q})$ Jacobien de force

k Rigidité de l'effecteur dans l'espace cartésien (modèle du système robot-environnement servant à la conception du contrôleur de force)

$K(s)$ Fonction de transfert générale du contrôleur de force

$\mathbf{K}$ Énergie cinétique du robot

k_0 Facteur d'ajustement de l'adaptation du gain intégral du contrôleur de force (limite inférieure)

k_1 Facteur d'ajustement de l'adaptation du gain intégral du contrôleur de force (Amplitude maximale)

k_2 Facteur d'ajustement de l'adaptation du gain intégral du contrôleur de force (sensibilité de variation en fonction de l'erreur de force)

$\mathbf{K_D}$ Matrice multipliant l'erreur de suivi **s** (gain du contrôleur PD)

k_e Rigidité effective de l'environnement

k_f Rigidité équivalente du système robot-environnement

k_i Gain intégral du contrôleur de force

K_i Énergie cinétique du membre *i* du robot

k_{lim} Limite supérieure (positive) du module de **P**

k_p Gain proportionnel du contrôleur de force

k_r Rigidité du ressort monté à une articulation rotative ou prismatique

k_{rs} Constante de rigidité modélisant l'impédance de l'environnement du robot

L Lagrangien du robot

L0 Longueur du membre 1 d'un doigt de la main de l'étude de cas

L1 Longueur du membre 2 d'un doigt de la main de l'étude de cas

L2 Longueur du membre 3 d'un doigt de la main de l'étude de cas

L3 Longueur du membre 4 d'un doigt de la main de l'étude de cas

L4 Longueur du membre 5 d'un doigt de la main de l'étude de cas

L_r Projection de la longueur du ressort sur l'axe des Z de l'articulation

m Masse de l'effecteur dans l'espace cartésien (modèle du système robot-environnement servant à la conception du contrôleur de force)

M Masse de l'objet

$\mathbf{M}$ Matrice des masses et inerties

m_i Masse du membre i

n Nombre d'articulations actives du robot

${}^{i}\mathbf{n}_{i}$ Couple exercé par le membre i-1 sur le membre i représenté dans le référentiel i

$\mathbf{N}_{i}$ Couple d'inertie agissant sur le membre i

n_p Nombre de processeurs utilisés pour le calcul en parallèle

p Nombre de degrés de liberté du robot

p Position calculée par le générateur de trajectoire

$\mathbf{P}$ Matrice de gain d'adaptation de l'identificateur (définie positive)

$\dot{\mathbf{P}}$ Dérivée de la matrice $\mathbf{P}$

p_0 Position initiale de la trajectoire de position

p_a Consigne de position filtrée calculée par le contrôleur de force

p_f Position finale de la trajectoire de position

${}^{i}\mathbf{P}_{Ci}$ Position de l'origine du référentiel de la masse du membre i par rapport au référentiel du membre i

${}^{0}\mathbf{P}_{Ci}$ Vecteur de position de la masse par rapport à la base du robot

P_{ex} Position de l'effecteur du doigt en x

${}^{i-1}\mathbf{P}_{i}$ Vecteur de position du référentiel i par rapport au référentiel $i-1$

p_x coordonnée en x de l'effecteur par rapport à la base du robot

p_y coordonnée en y de l'effecteur par rapport à la base du robot

p_z coordonnée en z de l'effecteur par rapport à la base du robot

P_{xo} Position en x de l'objet

P_{yo} Position en y de l'objet

px Position en x du centre de l'objet par rapport au référentiel de l'environnement

px_{1o} Position limite en x (axe du référentiel du doigt 1) de l'objet

px_{2o} Position limite en x (axe du référentiel du doigt 2) de l'objet

px_{3o} Position limite en x (axe du référentiel du doigt 3) de l'objet

py Position en y du centre de l'objet par rapport au référentiel de l'environnement

py_{1o} Position limite en y (axe du référentiel du doigt 1) de l'objet

py_{2o} Position limite en y (axe du référentiel du doigt 2) de l'objet

py_{3o} Position limite en y (axe du référentiel du doigt 3) de l'objet

$\mathbf{q}$ Vecteur des positions articulaires du robot

$\dot{\mathbf{q}}$ Vecteur des vitesses articulaires du robot

$\ddot{\mathbf{q}}$ Vecteur des accélérations articulaires du robot

$\tilde{\mathbf{q}}$ Vecteur des erreurs de positions articulaires

$\dot{\tilde{\mathbf{q}}}$ Vecteur des erreurs de vitesses articulaires

$\dot{\mathbf{q}}_d$ Vitesse de consigne du contrôleur de position

$\ddot{\mathbf{q}}_d$ Accélération de consigne du contrôleur de position

q_i Position articulaire du robot

$\dot{q}_i$ Vitesse articulaire du robot

$\ddot{q}_i$ Accélération articulaire du robot

$\dot{\mathbf{q}}_r$ Vitesse modifiée pour le retour d'état du contrôleur de position

$\dot{\tilde{\mathbf{q}}}_r$ Vecteur des erreurs de vitesse modifiée (**s**)

$\ddot{\mathbf{q}}_r$ Accélération modifiée pour le retour d'état du contrôleur de position

$\mathbf{R}$ Matrice de poids d'importance de l'information donnée par l'erreur des paramètres estimés pour la loi d'adaptation

${}^{i-1}_{i}\mathbf{R}$ Matrice de rotation reliant le référentiel i au référentiel $i-1$

s Opérateur de Laplace

$\mathbf{s}$ Erreur de suivi du contrôleur de position

t Temps

${}^{i-1}_{i}\mathbf{T}$ Matrice de transformation homogène du référentiel i vers le référentiel i-1

t_1 Temps de calcul en séquentiel

t_f Temps final (durée) de la trajectoire

t_p Temps de calcul en parallèle

$\mathbf{U}$ Énergie potentielle du robot

U_i Énergie potentielle du membre i du robot

U_{ref} Énergie potentielle de référence choisie pour que la valeur minimum de U_i soit nulle

v Vitesse calculée par le générateur de trajectoire

$\vec{\mathbf{v}}$ Vecteur unitaire possédant la même orientation que la gravité

v_a Consigne de vitesse filtrée calculée par le contrôleur de force

${}^{i}\dot{\mathbf{v}}_{Ci}$ Accélération de l'origine du référentiel de la masse du membre i par rapport au référentiel du membre i

v_f Vitesse désirée de l'effecteur

$^{i+1}v_{i+1}$ Vitesse prismatique du membre i+1 par rapport à la base

$^{i+1}\dot{v}_{i+1}$ Accélération prismatique du membre i+1 par rapport à la base

v_x Vitesse galliléenne linéaire de l'effecteur en x par rapport au référentiel de la base du robot

v_y Vitesse galliléenne linéaire de l'effecteur en y par rapport au référentiel de la base du robot

v_z Vitesse galliléenne linéaire de l'effecteur en z par rapport au référentiel de la base du robot

w Filtre permettant le calcul de **W** sans accélération mesurée

w Position de l'objet (w mesuré dans la direction x ou y)

W Matrice de signaux de l'identificateur

w_x Vitesse galliléenne rotative de l'effecteur en x par rapport au référentiel de la base du robot

w_y Vitesse galliléenne rotative de l'effecteur en y par rapport au référentiel de la base du robot

w_z Vitesse galliléenne rotative de l'effecteur en z par rapport au référentiel de la base du robot

W00 Première partie de la matrice **W** devant être filtrée par w

W01 Seconde partie de la matrice **W** devant être dérivée et filtrée par w

W02 Troisième partie de la matrice **W**

$\dot{\mathbf{X}}$ Vecteur des vitesses galliléennes linéaires et rotatives

$x_f(s)$ Position désirée de l'effecteur (consigne calculée par le contrôleur de force)

$Y(s)$ Admittance du contrôleur de force

$\mathbf{Y}$ Matrice régresseur du robot (contrôleur de position et identificateur)

${}^{i+1}\hat{\mathbf{Z}}_{i+1}$ Vecteur de transformation d'un scalaire en vecteur orienté vers l'axe des Z et de même amplitude que le scalaire

α_{i-1} Angle entre Z_{i-1} et Z_i autour de X_{i-1}

Δx Perturbation de la position de l'effecteur

Δx_c Déplacement désiré de l'effecteur

Δx_r Consigne de position lors d'une commande en position et perturbation de position lors d'une consigne de force

η Facteur de convergence du contrôleur de position ($1/\eta$ est la constante de temps de convergence vers zéro des l'erreurs de vitesse et de position du contrôleur de position)

θ_{123} Somme des trois angles d'un doigt de l'étude de cas

θ_e Orientation de l'effecteur par rapport à la base

θ_i Angle entre X_{i-1} à X_i autour de Z_i

λ_i Coordonnées généralisées du robot

$\mathbf{\Lambda}$ Matrice constante avec valeurs propres dans le demi-plan droit définissant la rigidité du contrôleur de position

ξ_i Forces généralisées associées aux coordonnées généralisées λ_i

ρ_0 Facteur d'oubli de l'identificateur (valeur entre 0 et 1 ; près de 0 : peu d'oubli, près de 1 : beaucoup d'oubli)

$\boldsymbol{\tau}$ Vecteur de forces/couple articulaires

$\hat{\boldsymbol{\tau}}_f$ Couple observé filtré (calculé à partir de $\mathbf{W}$ et $\hat{\mathbf{a}}$)

$\boldsymbol{\tau}_f$ Couple de commande filtré par w

$\boldsymbol{\tau}_i$ Couple articulaire (articulation i)

φ_x Angles définissant l'orientation de l'effecteur autour de l'axe x du référentiel de la base du robot

φ_y Angles définissant l'orientation de l'effecteur autour de l'axe x du référentiel de la base du robot

φ_z Angles définissant l'orientation de l'effecteur autour de l'axe x du référentiel de la base du robot

${}^{i+1}\boldsymbol{\omega}_{i+1}$ Vitesse angulaire du membre i+1 par rapport à la base

${}^{i+1}\dot{\boldsymbol{\omega}}_{i+1}$ Accélération angulaire du membre i+1 par rapport à la base

CHAPITRE I

INTRODUCTION

1.1. Modélisation, commande et simulation de robots

La modélisation de systèmes robotiques demande l'utilisation d'algorithmes permettant de calculer de façon efficace les équations décrivant la cinématique et la dynamique d'un robot. Le modèle de la cinématique directe d'un robot est une représentation géométrique de sa structure. Il s'agit de calculer la position et l'orientation de l'effecteur en fonction des variables articulaires du robot. Le modèle de la cinématique inverse d'un robot peut être utilisé pour convertir une trajectoire dans le domaine galliléen (espace de travail) en trajectoire dans le domaine articulaire. La planification de tâches et d'assemblage sont deux applications courantes en robotique requérant le modèle de la cinématique directe et une détection de collision (ex : [1]). Les paramètres Denavit-Hartenberg modifiés, comme montrés par Craig [2], sont très répandus comme convention pour la modélisation de la cinématique directe. Avec le modèle cinématique, nous pouvons calculer la dynamique du robot et évaluer son comportement.

Le développement de lois de commande et la planification d'assemblage pour les robots avec éléments flexibles requièrent le modèle dynamique (ex : [3], [4]). La formulation du modèle dynamique est l'élément clef contrôlant l'efficacité et la flexibilité de l'algorithme de simulation du robot. Le modèle de la dynamique directe d'un robot permet de calculer la position, vitesse et accélération articulaires en fonction du couple appliqué aux articulations. Le modèle de la dynamique inverse peut servir de base à la mise en œuvre de divers contrôleurs. Les méthodes de Lagrange-Euler et Newton-Euler

sont les plus répandues pour modéliser la dynamique de manipulateurs robotiques (ex : [5]). Les équations de D'Alembert représentent l'autre méthode principale pour trouver le modèle dynamique du robot (ex : [6]).

Il est souvent long et ardu de calculer les équations des modèles cinématiques et dynamiques et de les écrire dans un langage de programmation pour la simulation. C'est pour cette raison que plusieurs chercheurs ont développé des outils permettant de générer automatiquement le code de simulation à partir d'une description du robot. Khalil [7] présente un générateur automatique des équations symboliques de la cinématique, de la dynamique et d'un identificateur de paramètres de robots. Chen [8] propose un outil de génération automatique du code de simulation du modèle dynamique pour un robot à deux articulations. Chen et Yang [9] présentent un outil de génération automatique de modèle de robot en arbre à l'aide d'une matrice de connectivité. Finalement, Piedboeuf [10] présente l'outil SYMOFROS pour la modélisation sur Maple™ et la génération automatique de code pour la simulation de manipulateurs flexibles. Des outils plus généraux (ex : Dymola [11]) offrent les fonctionnalités suivantes : modélisation et simulation de modèles cinématiques et dynamiques de robots de topologie série avec boucles cinématiques fermées, utilisation étendue de manipulations symboliques, interface graphique, gestion d'événement pour la simulation en temps réel, utilisation des modèles de robots pour la conception de contrôleurs et compatibilité avec de multiples plate-formes.

Les tâches les plus courantes en robotique sont la manipulation, l'insertion, l'ébavurage etc. Pour effectuer ces tâches, nous devons commander le contact entre le manipulateur robotique et son environnement. Ce besoin a mené à la conception de plusieurs stratégies de commande position/force. Le système à commander dans ce cas est l'ensemble robot-environnement-contrôleur de position ; il faut supposer que la commande de position est suffisamment précise pour nos besoins. Certaines tâches plus complexes demandent la collaboration de plusieurs robots (ex : [12]). Dans tous les cas, nous désirons commander les forces de contact avec précision pendant un contact.

Whitney [13] présente l'algorithme "position accomodation" permettant de commander la force de contact en modifiant la trajectoire de position en fonction de la force captée. Raibert et Craig [14] introduisent le concept de commande hybride. Cette approche permet de commander l'effecteur en position pour les directions sans contraintes et en force pour les directions en contact. Hogan [15] propose la commande par impédance, permettant de définir le comportement de la position de l'effecteur en fonction de la force de contact. Il arrive que les paramètres connus d'un système robotique soient erronés. Un problème pouvant être rencontré dans le cas d'un contrôleur mal ajusté est la présence de rebondissements de l'effecteur au moment du contact entre l'effecteur avec un objet. Ce problème peut être minimisé en utilisant un contrôleur s'adaptant aux changements des conditions d'opérations. Par exemple, pour un système multi-robot tel que le robot de service de la navette spatiale, la masse des objets manipulés peut varier et l'impédance mécanique des objets manipulés en commun est incertaine. Lin et Yae [16] présentent un algorithme d'identification des paramètres de l'environnement du robot. L'algorithme

permet d'identifier les frontières physiques et la rigidité de l'objet saisi. Seraji [17] présente des lois de commande adaptatives et non linéaires ayant l'avantage d'augmenter la robustesse du contrôleur de force face aux changements des paramètres de l'environnement. Ces algorithmes permettent soit d'identifier la rigidité de l'objet saisi, soit d'adapter le contrôleur au comportement de l'environnement.

La plupart des algorithmes de commande position/force demandent que la surface de l'objet à saisir soit flexible. De plus, ces travaux posent comme hypothèse que l'impédance de l'objet à saisir est principalement caractérisée par la rigidité de l'objet. Vukobravic [18] étudie le cas où la dynamique de l'objet contacté n'est pas négligeable. Pandian et Kawamura [19] décrivent des algorithmes permettant de commander un robot en contact avec une surface rigide.

Les algorithmes de commande de force requièrent toujours un algorithme de commande de position fonctionnant simultanément pour commander le manipulateur dans les directions non contraintes. Il existe beaucoup d'algorithmes de commande de position pouvant remplir cette fonction. L'algorithme développée par Slotine et Li [20] comprend un contrôleur de position articulaire. Un des avantages du contrôleur est qu'il n'a pas besoin de la mesure d'accélération. Il comprend aussi un identificateur permettant d'estimer les paramètres dynamiques d'un robot rigide en mouvement libre. Cet algorithme contient aussi un facteur d'oubli lui permettant de mettre les paramètres observés à jour si les paramètres réels varient.

De plus, la commande automatisée de robots effectuant des tâches de contact demande l'utilisation d'un algorithme de supervision. Cet algorithme doit contrôler les trajectoires demandées au robot et gérer les contrôleurs et identificateurs. Par exemple, nous ne devons pas tenter d'identifier les masses du robot pendant un contact. Cette action aurait comme conséquence de faire diverger les paramètres identifiés.

La simulation en temps réel des algorithmes cités précédemment requiert souvent trop de puissance de calcul pour les ordinateurs de type PC actuels. La plupart des travaux traitant de calcul en parallèle et en temps réel des modèles robotiques utilisent des architectures dédiées. Par exemple, Zhang et al. [21] présentent une formulation de Newton-Euler de la dynamique inverse qui est appropriée au calcul en parallèle. Par contre, ils ne tiennent pas compte des temps de communication. Abdalla [22] montre le parallélisme inhérent de l'algorithme de Newton-Euler dans le but de calculer la dynamique inverse et directe. Pu et al. [23] proposent une méthode de calcul de la matrice des masses et inerties pour un robot en forme d'arbre. Cette méthode est optimisée pour le calcul en parallèle à faible grain. Elle n'est donc pas adaptée à un système de simulation informatique général avec peu de nœuds de calculs.

L'apparition d'outils logiciels et matériels multi-PC sur le marché rend accessible financièrement la simulation en parallèle. Opal-rt Technologies inc. [24] présente un outil permettant de séparer le code de simulation séquentiel pour la simulation en parallèle. L'avantage d'un tel système par rapport à un réseau de processeurs dédiés est la facilité avec laquelle nous pouvons changer le programme exécuté. Le système multi-

PC est donc très intéressant pour le domaine de la recherche, du développement et de l'éducation. Les articles traitant de simulation en parallèle de robots utilisent des architectures à faible grain (et donc beaucoup de communication). Dans ce cas, la séparation du code se situe au niveau algorithmique. Avec un système multi-PC, il est plus avantageux de séparer le code au niveau fonctionnel.

1.2. Objectifs

Nous désirons développer un outil convivial permettant de concevoir rapidement et simplement des modèles de robots rigides ainsi que leurs contrôleurs. L'automatisation de l'écriture du code de simulation permet d'accélérer le processus de conception de systèmes robotiques. L'outil doit être flexible d'utilisation. Par exemple, il doit avoir la possibilité d'inclure des modèles de systèmes d'entraînement. Les modèles conçus doivent être performants en terme de temps de calcul pour permettre la simulation en temps réel. Pour atteindre de bonnes performances en temps réel, nous séparons et exécutons notre code sur une plate-forme multiprocesseurs. Les capacités de l'outil doivent être démontrées avec un exemple exigeant, représentatif d'applications réelles du point de vue du modèle du robot et de la commande. L'objectif de commande pour la simulation d'un système multi-robots en contact avec l'environnement est d'obtenir des erreurs de force et de position minimales. Des algorithmes et une stratégie de commande doivent être développés pour obtenir un régime transitoire de début de contact et de relâchement d'un objet minimal en temps et en amplitude. De plus, le contrôleur de position/force doit être robuste aux changements des paramètres du robot et de

l'environnement. Par exemple, le contrôleur de force doit s'adapter à une rigidité de l'environnement inconnue.

1.3. Méthode

Nous concevons un générateur de code produisant automatiquement les fichiers de simulation à partir d'une brève description du robot. La méthode de travail utilisée pour mener à bien ce projet est présentée à la figure 1.1.

Nous utilisons Simulink™ comme plate-forme de simulation pour des raisons de compatibilité avec une multitude de logiciels. Par exemple, à l'aide de *Real-Time Workshop* de *Matlab*® et du séparateur de code de Opal-rt Technologies inc. [24], nous pouvons facilement et rapidement simuler nos codes en parallèle sur un système multi-PC. De plus, il existe des outils (ex : Adams et Sense8) pouvant s'interfacer à *Simulink*™ et permettant de visualiser en 3 dimensions le robot et ce en temps réel ou différé. Nous avons choisi le langage C++ pour l'écriture de nos algorithmes. Ce choix est avantageux du point de vue de la transportabilité et de l'efficacité des compilateurs C++ actuels ; le compilateur utilisé est *Watcom C++ 11.0*.

Le générateur automatique de code utilise les algorithmes de Newton-Euler et Lagrange-Euler. Ces algorithmes sont les plus utilisés dans le domaine de la robotique. Ils sont efficaces pour modéliser des robots de topologie en série ou en arbre. Ferretti [25] et Megahed [26] exposent des méthodes de modélisation de robots plus complexes

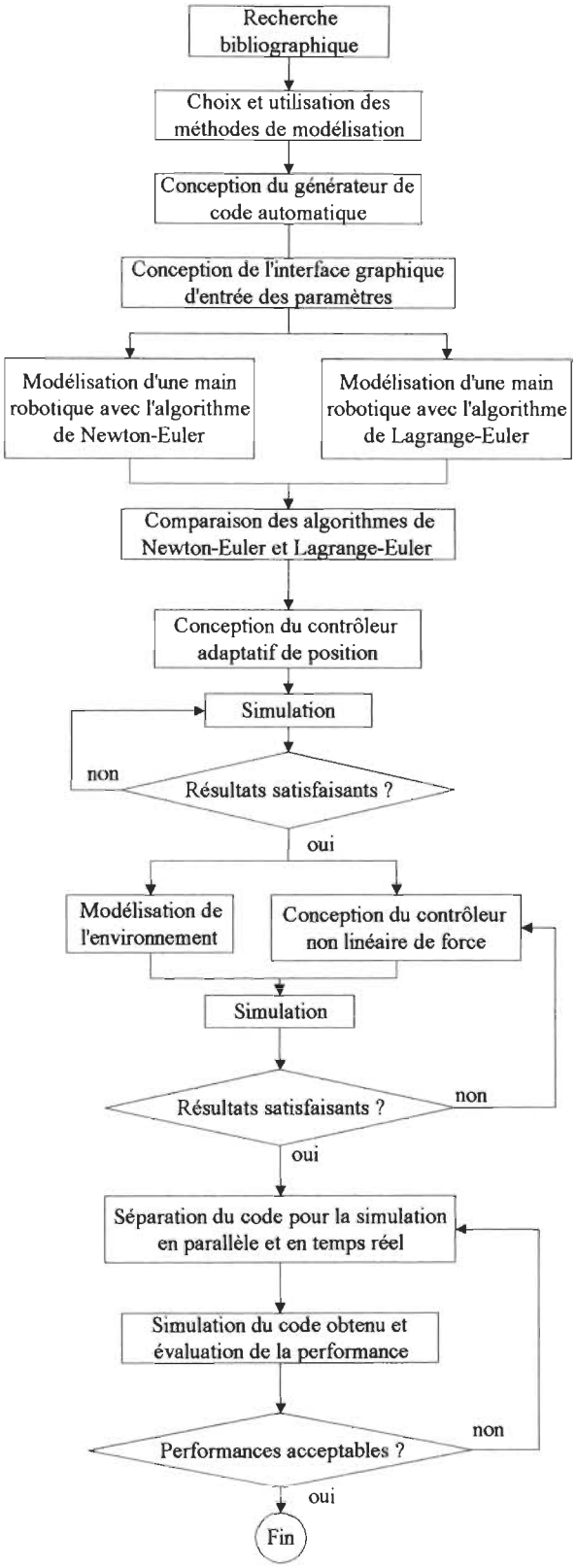

Figure 1-1 : Méthode de travail

contenant des boucles cinématiques fermées. Entre autre, ils utilisent le multiplicateur de Lagrange et une méthode de résolution explicite des équations de contraintes.

L'algorithme de Newton-Euler est simple à utiliser, rapide à exécuter et sa complexité croit linéairement en fonction du nombre d'articulations. L'algorithme de Lagrange-Euler est plus rapide à exécuter que Newton-Euler pour un nombre réduit d'articulations mais sa complexité croit rapidement pour un nombre plus important d'articulations.

Nous utilisons le Extended *Symbolic Toolbox* de *Matlab*® pour calculer les équations des modèles de robots sous forme symbolique. Ces équations sont écrites dans des fichiers texte à l'aide de *Matlab*® et *Perl*. Nous constituons une banque de données contenant la structure principale des sous-routines produites par le générateur de code. C'est à l'aide de cette banque de données et des équations écrites dans des fichiers textes que le générateur doit produire le code final de simulation. La structure des blocs *Simulink*™ de simulation du modèle de Lagrange-Euler est toujours la même pour un robot de topologie série. Par contre, lorsque le générateur de code utilise la formulation de Newton-Euler, la structure est fonction du nombre d'articulations. Le générateur de code doit donc produire deux librairies : une pour les blocs de calcul de la cinématique directe articulaire et une pour les blocs de calcul de la dynamique inverse articulaire. L'utilisateur devra par la suite produire lui-même le modèle complet du robot à l'aide de ces blocs et de la librairie de calcul matriciel et robotique (produit de matrices et de vecteurs, fonctions nécessaires à la dynamique inverse sous la forme de Newton-Euler, etc.). Le générateur de code produit ensuite un contrôleur de position en fonction du robot à commander. En

fait, il utilise la matrice régresseur de la dynamique inverse du modèle du système pour produire le contrôleur de position et l'identificateur de paramètres de Slotine et Li. Nous validons le générateur de code en comparant le couple obtenu avec l'algorithme de Newton-Euler avec celui obtenu avec l'algorithme de Lagrange-Euler.

Nous évaluons aussi le générateur de code en modélisant une main robotique en contact avec son environnement et en effectuant plusieurs simulations de saisie d'un objet. Les lois de commande sont évaluées pour plusieurs rigidités des surfaces de l'objet saisi et en fonction d'erreurs de modélisation cinématique. Nous utilisons le contrôleur et l'identificateur présentés par Slotine et Li et le contrôleur non linéaire de force présenté par Seraji (avec modèle d'impédance de premier ordre) pour commander la force de contact de notre manipulateur en tout temps.

Nous produisons une fonction *Matlab*® permettant d'afficher en 3 dimensions le robot. De cette façon, nous obtenons un affichage de la position du robot sans avoir à acheter un autre logiciel de réalité virtuelle.

Pour simuler un système robotique en temps réel avec un pas de calcul suffisamment petit, nous utilisons un système multi-PC caractérisé par des liens de communication à haute vitesse. Nous séparons notre code séquentiel pour obtenir les codes de simulation en parallèle pour le système multi-PC à l'aide du logiciel RT-Lab de Opal-rt [24]. La séparation est effectuée au niveau fonctionnel, c'est à dire qu'un nœud de calcul peut par exemple contenir le modèle d'un doigt complet avec ses contrôleurs. Le système Opal-rt

utilise des liens de communication à haute vitesse IEEE 1394 200Mbit/s. Les nœuds de calcul sont des Pentium II 233 avec 16Mo de RAM et sont configurés avec QNX Real-Time Operating System. La structure est totalement connectée et elle est constituée d'un nœud maître et d'un nombre quelconque de nœuds esclaves. Le système comprend aussi un nœud console configuré sous Windows-NT. Ce nœud permet d'enregistrer les données de simulation et de visualiser le robot en 3 dimensions. Après avoir évalué l'efficacité de la parallélisation de l'algorithme sur le système Opal-rt, nous comparons les performances des algorithmes de Lagrange-Euler et Newton-Euler du point de vue du temps de calcul.

1.4. Structure du mémoire

Le chapitre II décrit les algorithmes de modélisation cinématique et dynamique employés tout au long de ce travail. Nous présentons la méthode de modélisation cinématique par les paramètres Denavit-Hartenberg modifiés. Ensuite, nous exposons les algorithmes de dynamique inverse de Newton-Euler et Lagrange-Euler. Finalement, nous présentons deux méthodes de calcul de la dynamique directe.

Le chapitre III décrit les contrôleurs et le superviseur utilisés dans ce travail. Nous expliquons le fonctionnement du contrôleur adaptatif de position de Slotine et Li et du contrôleur de force de Seraji. Nous exposons ensuite les stratégies implantées dans le bloc superviseur.

Le chapitre IV décrit le générateur automatique de code. Nous expliquons son fonctionnement par rapport à la formulation choisie, c'est à dire Newton-Euler ou Lagrange-Euler. Nous montrons ensuite les stratégies de conception des lois de commande à partir des équations du modèle généré automatiquement. Finalement, nous exposons les programmes d'initialisation et la librairie de calcul matriciel et de robotique.

Le chapitre V constitue notre étude de cas. Nous commençons par une description du manipulateur, de son environnement et du programme de visualisation du robot en 3 dimensions. Ensuite, nous voyons les schémas *Simulink*™ du modèle du manipulateur sous la forme de Lagrange-Euler. Nous montrons aussi une partie des blocs de simulation du modèle sous la formulation de Newton-Euler. Après une description des essais effectués, nous pouvons constater l'efficacité des algorithmes conçus en visualisant les résultats de simulation en séquentiel. Nous évaluons ensuite l'efficacité de la simulation en parallèle et montrons qu'il est possible d'atteindre le temps réel avec une précision satisfaisante. Finalement, nous comparons le modèle sous la formulation de Newton-Euler avec celui sous la formulation de Lagrange-Euler du point de vue du temps de calcul.

Le chapitre VI conclut le mémoire en présentant les contributions et les recommandations.

CHAPITRE II

MODÉLISATION CINÉMATIQUE ET DYNAMIQUE

La modélisation de systèmes robotiques demande l'utilisation d'algorithmes permettant de calculer de façon efficace les équations décrivant la cinématique et la dynamique d'un robot. Ces mêmes équations peuvent aussi servir à produire les contrôleurs du robot et le modèle de l'objet entrant en contact avec ce robot.

Les modèles des robots produits à l'aide des algorithmes de modélisation doivent être flexibles et structurés. C'est à dire que nous devons pouvoir y ajouter facilement de l'information obtenue de différentes sources : limites, phénomènes observés expérimentalement, non linéarités etc.

Nous introduisons la modélisation cinématique employant des matrices de transformation homogènes et utilisant les paramètres Denavit-Hartenberg modifiés. Nous calculons les modèles de la dynamique de robots à partir des équations de Newton-Euler et Lagrange-Euler.

2.1. Matrice de transformation avec convention D-H modifiée

Nous désirons représenter la position et l'orientation d'un organe effecteur (pince, outil, etc.) par rapport à un référentiel de base. Pour cela, il nous faut déterminer les transformations qui relient chaque référentiel au précédent. Dans le cas d'un manipulateur à membres rigides, seulement 4 paramètres sont nécessaires pour décrire

complètement chaque membre. Ces paramètres, suite au choix de la position des référentiels, permettent d'obtenir la position et l'orientation des référentiels par rapport aux autres. Cependant, pour les membres prismatiques et rotatifs, trois des paramètres sont fixes et un seul est variable. Nous devons choisir la position des référentiels en tenant compte de certaines contraintes. Posons que la structure d'un membre est représentée par le référentiel « i ». Si l'articulation « i » est prismatique, alors l'axe Z_i doit être parallèle à l'axe de translation de l'articulation. Si l'articulation est rotative, alors l'axe Z_i doit être placé sur l'axe de rotation du membre. De plus, il est préférable d'orienter l'axe X_i vers le prochain membre. L'axe Y_i est choisi pour compléter le système de coordonnées selon la règle de la main droite. Avec la représentation Denavit-Hartenberg modifiée, ces 4 paramètres sont représentés à la figure 2.1 et sont définis de la façon suivante [5]: a_{i-1} est la distance de Z_{i-1} à Z_i mesurée le long de X_{i-1}, α_{i-1} est l'angle entre Z_{i-1} et Z_i autour de X_{i-1}, d_i est la distance de X_{i-1} à X_i mesurée le long de Z_i et θ_i est l'angle entre X_{i-1} à X_i autour de Z_i .

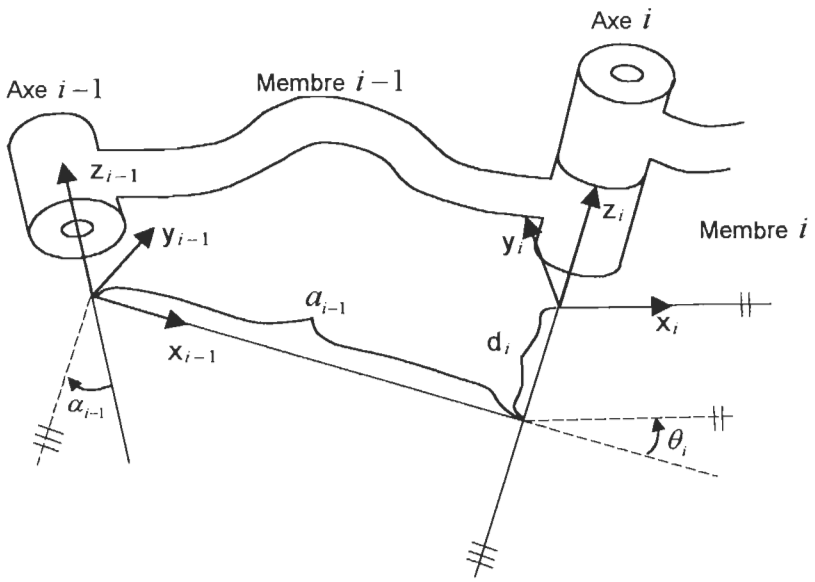

Figure 2-1 : Configuration d'un membre en fonction des paramètres D-H modifiés

Connaissant la configuration de chacune des articulations, nous pouvons produire les matrices de transformation homogènes à partir de ce modèle général:

$$ {}^{i-1}_{i}\mathbf{T} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\ \sin(\theta_i)\cos(\alpha_{i-1}) & \cos(\theta_i)\cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -\sin(\alpha_{i-1})d_i \\ \sin(\theta_i)\sin(\alpha_{i-1}) & \cos(\theta_i)\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{c|c} {}^{i-1}_{i}\mathbf{R} & {}^{i-1}\mathbf{P}_i \\ \hline 0\,0\,0 & 1 \end{array}\right] \quad (2.1) $$

où ${}^{i-1}_{i}\mathbf{T}$ est la matrice de transformation homogène du référentiel i vers le référentiel i-1, ou la représentation du référentiel i dans $i-1$ en terme d'orientation et de position.

La matrice carrée ${}^{i-1}_{i}\mathbf{R}$ est la matrice de rotation reliant le référentiel i au référentiel $i-1$. Le vecteur ${}^{i-1}\mathbf{P}_i$ est le vecteur de position du référentiel i par rapport au référentiel $i-1$. La quatrième ligne peut être employée pour effectuer une transformation homogène de perspective et d'échelle. Ces transformations ne sont pas employées; le facteur d'échelle est unitaire et le vecteur de transformation de perspective est [0 0 0].

Finalement, pour obtenir la représentation d'un membre par rapport à la base, il suffit de multiplier ensemble les matrices de transformation homogènes ${}^{0}_{1}\mathbf{T}\ {}^{1}_{2}\mathbf{T} \ldots {}^{n-1}_{n}\mathbf{T}$ où n est le numéro de l'articulation. Le résultat est la position galliléenne (position et orientation) du membre en fonction des variables articulaires; nous avons donc effectué la cinématique directe du manipulateur.

Notons qu'avec cette notation, une articulation peut être rotative (avec d, α et a constants et θ variable) ou prismatique (avec d variable et θ, α et a constants).

La cinématique inverse permet de calculer les positions articulaires en fonction des positions galliléennes (dans l'espace de travail). Elle peut être obtenue par des méthodes géométriques ou itératives. Il arrive souvent qu'il y ait de multiples solutions à la cinématique inverse d'un manipulateur. Dans ce cas, nous devons effectuer des choix en fonction de la géométrie du manipulateur, ce qui rend la génération automatique de la cinématique inverse difficile.

Le Jacobien de vitesse permet de calculer les vitesses galliléennes en fonction des vitesses articulaires :

$$\dot{\mathbf{X}} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ w_x \\ w_y \\ w_z \end{bmatrix} = \mathbf{J(q)} \begin{bmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_n \end{bmatrix} \tag{2.2}$$

où v_x, v_y et v_z sont les vitesses galliléennes linéaires de l'effecteur respectivement en x, y et en z par rapport au référentiel de la base du robot. Les variables w_x, w_y et w_z sont les vitesses galliléennes rotatives de l'effecteur respectivement en x, y et en z par rapport au référentiel de la base. Les variables $\dot{q}_1$ à $\dot{q}_n$ sont les vitesses articulaires. Pour calculer le Jacobien de vitesse, nous utilisons :

$$\mathbf{J(q)} = \begin{bmatrix} \partial p_x / \delta q_1 & \partial p_x / \delta q_2 & \cdots & \partial p_x / \delta q_n \\ \partial p_y / \delta q_1 & \partial p_y / \delta q_2 & \cdots & \partial p_y / \delta q_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial \phi_z / \delta q_1 & \partial \phi_z / \delta q_2 & \cdots & \partial \phi_z / \delta q_n \end{bmatrix} \tag{2.3}$$

où p_x, p_y et p_z sont les coordonnées de l'effecteur par rapport à la base. φ_x, φ_y et φ_z sont les angles définissant l'orientation de l'effecteur autour des axes x, y et z du référentiel de la base.

Soit n le nombre d'articulations du robot et p le nombre de degrés de liberté. Pour calculer les vitesses articulaires du manipulateur en fonction des vitesses galliléennes, nous devons utiliser le Jacobien de vitesse inverse comme suit.

$$\dot{\mathbf{q}} = \mathbf{J(q)}^{-1} \dot{\mathbf{X}} \tag{2.4}$$

Si $n = p$, alors nous pouvons inverser directement le Jacobien de vitesse. Si $n < p$ alors l'inverse du Jacobien peut être calculé par

$$\mathbf{J}^* = \left(\mathbf{J}^T(\mathbf{q})\mathbf{J}(\mathbf{q})\right)^{-1} \mathbf{J}(\mathbf{q})^T \tag{2.5}$$

Par contre, si $n > p$ alors l'inverse généralisé est calculé par

$$\mathbf{J}^* = \mathbf{J}(\mathbf{q})^T \left(\mathbf{J}(\mathbf{q})\mathbf{J}(\mathbf{q})^T\right)^{-1} \tag{2.6}$$

Nous pouvons calculer l'accélération articulaire en fonction de l'accélération galliléenne par

$$\ddot{\mathbf{q}} = \dot{\mathbf{J}}^{-1}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{X}} + \mathbf{J}^{-1}(\mathbf{q})\ddot{\mathbf{X}} \tag{2.7}$$

2.2. Dynamique inverse

L'opération de la dynamique inverse consiste à calculer le couple résultant à l'articulation à partir des variables articulaires (positions, vitesses et accélérations). Le résultat est utile pour calculer la dynamique directe (modèle de robot) et certains contrôleurs (commande en avance, linéarisation par retour d'état etc.).

2.2.1. Algorithme de Newton-Euler

Nous allons maintenant voir les équations de vitesse, d'accélération, de force et de couple nous permettant d'obtenir la dynamique inverse. Tous les calculs qui suivent sont effectués selon la méthode de Newton-Euler, de façon itérative, i.e. membre par membre. La première partie des itérations se fait de la base vers l'effecteur ; pour calculer les forces et couples d'inertie d'un certain membre, nous avons besoin des résultats du membre précédent. La seconde partie de l'algorithme, qui consiste à calculer les forces et couples articulaires, s'effectue de l'effecteur à la base. Ces formules sont démontrées dans [2], [5].

D'abord, nous devons calculer les vitesses angulaire et linéaire de chacun des membres. La notation ${}^{i+1}\boldsymbol{\omega}_{i+1}$ signifie la vitesse angulaire du membre i+1 par rapport à la base, représentée dans le référentiel i+1. Le vecteur ${}^{i+1}\hat{\mathbf{Z}}_{i+1} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ sert à transformer un

scalaire en vecteur orienté vers l'axe des Z. Pour une articulation rotative, les formules sont les suivantes :

$$ {}^{i+1}\boldsymbol{\omega}_{i+1} = {}^{i+1}_{i}\mathbf{R}\ {}^{i}\boldsymbol{\omega}_{i} + \dot{\boldsymbol{\theta}}_{i+1}\ {}^{i+1}\hat{\mathbf{Z}}_{i+1} \tag{2.8} $$

$$ {}^{i+1}v_{i+1} = {}^{i+1}_{i}\mathbf{R}\left({}^{i}\mathbf{v}_{i} + {}^{i}\boldsymbol{\omega}_{i} \times {}^{i}\mathbf{P}_{i+1}\right) \tag{2.9} $$

Pour une articulation prismatique, les équations de vitesses rotative et linéaire sont :

$$ {}^{i+1}\boldsymbol{\omega}_{i+1} = {}^{i+1}_{i}\mathbf{R}\ {}^{i}\boldsymbol{\omega}_{i} \tag{2.10} $$

$$ {}^{i+1}\mathbf{v}_{i+1} = {}^{i+1}_{i}\mathbf{R}\left({}^{i}\mathbf{v}_{i} + {}^{i}\boldsymbol{\omega}_{i} \times {}^{i}\mathbf{P}_{i+1}\right) + \dot{\mathbf{d}}_{i+1}\ {}^{i+1}\hat{\mathbf{Z}}_{i+1} \tag{2.11} $$

Ensuite nous devons calculer les accélérations rotative et linéaire de l'origine de chacun des référentiels des membres. Pour une articulation rotative, les formules sont les suivantes :

$$ {}^{i+1}\dot{\boldsymbol{\omega}}_{i+1} = {}^{i+1}_{i}\mathbf{R}\ {}^{i}\dot{\boldsymbol{\omega}}_{i} + {}^{i+1}_{i}\mathbf{R}\ {}^{i}\boldsymbol{\omega}_{i} \times \dot{\boldsymbol{\theta}}_{i+1}\ {}^{i+1}\hat{\mathbf{Z}}_{i+1} + \ddot{\boldsymbol{\theta}}_{i+1}\ {}^{i+1}\hat{\mathbf{Z}}_{i+1} \tag{2.12} $$

$$ {}^{i+1}\dot{\mathbf{v}}_{i+1} = {}^{i+1}_{i}\mathbf{R}\left({}^{i}\dot{\boldsymbol{\omega}}_{i} \times {}^{i}\mathbf{P}_{i+1} + {}^{i}\boldsymbol{\omega}_{i} \times \left({}^{i}\boldsymbol{\omega}_{i} \times {}^{i}\mathbf{P}_{i+1}\right) + {}^{i}\dot{\mathbf{v}}_{i}\right) \tag{2.13} $$

Pour une articulation prismatique, les équations d'accélérations rotative et linéaire de chaque membre sont :

$$ {}^{i+1}\dot{\boldsymbol{\omega}}_{i+1} = {}^{i+1}_{i}\mathbf{R}\ {}^{i}\dot{\boldsymbol{\omega}}_{i} \tag{2.14} $$

$$ {}^{i+1}\dot{\mathbf{v}}_{i+1} = {}^{i+1}_{i}\mathbf{R}\left({}^{i}\dot{\boldsymbol{\omega}}_{i} \times {}^{i}\mathbf{P}_{i+1} + {}^{i}\boldsymbol{\omega}_{i} \times \left({}^{i}\boldsymbol{\omega}_{i} \times {}^{i}\mathbf{P}_{i+1}\right) + {}^{i}\dot{\mathbf{v}}_{i}\right) \tag{2.15} $$

$$ + 2\ {}^{i+1}\boldsymbol{\omega}_{i+1} \times \dot{\mathbf{d}}_{i+1}\ {}^{i+1}\hat{\mathbf{Z}}_{i+1} + \ddot{\mathbf{d}}_{i+1}\ {}^{i+1}\hat{\mathbf{Z}}_{i+1} $$

Nous devons aussi trouver l'accélération linéaire du centre de masse de chaque membre. Nous posons qu'un référentiel « Ci » est attaché à chaque lien avec son origine positionnée au centre de masse du membre et avec la même orientation que le référentiel « i ». L'équation suivante est valide pour les articulations rotatives et linéaires :

$$ {}^{i}\dot{\mathbf{v}}_{Ci} = {}^{i}\dot{\boldsymbol{\omega}}_{i} \times {}^{i}\mathbf{P}_{Ci} + {}^{i}\boldsymbol{\omega}_{i} \times \left({}^{i}\boldsymbol{\omega}_{i} \times {}^{i}\mathbf{P}_{Ci} \right) + {}^{i}\dot{\mathbf{v}}_{i} \tag{2.16} $$

où ${}^{i}\mathbf{P}_{Ci}$ et ${}^{i}\dot{\mathbf{v}}_{Ci}$ sont la position et l'accélération de l'origine du référentiel de la masse du membre i par rapport au référentiel du membre i. Nous sommes maintenant en mesure d'appliquer les équations de Newton-Euler pour calculer les forces $\mathbf{F}_i$ et couples $\mathbf{N}_i$ d'inertie agissant sur chaque membre :

$$ \mathbf{F}_{i} = m_{i}\ \dot{\mathbf{v}}_{Ci} \tag{2.17} $$

$$ \mathbf{N}_{i} = {}^{Ci}\mathbf{I}_{i}\ {}^{i}\dot{\boldsymbol{\omega}}_{i} + {}^{i}\boldsymbol{\omega}_{i} \times {}_{Ci}\mathbf{I}\ {}^{i}\boldsymbol{\omega}_{i} \tag{2.18} $$

où ${}^{Ci}\mathbf{I}_{i}$ est le tenseur d'inertie au centre de masse représenté dans le référentiel i et définissant la forme et la distribution de la masse du membre.

Une fois toutes les vitesses, accélérations, forces et couples calculés pour chaque membre (de la base à l'effecteur), nous devons effectuer un calcul itératif dans le sens inverse (partant de l'effecteur vers la base). Nous calculerons les forces et couples aux joints qui résultent des forces et couples d'inertie agissant sur chaque membre et des forces et couples provenant des membres suivants (forces et couples externes) :

$$ {}^{i}\mathbf{f}_{i} = {}^{i}_{i+1}\mathbf{R}\ {}^{i+1}\mathbf{f}_{i+1} + {}^{i}\mathbf{F}_{i} \tag{2.19} $$

$$ {}^{i}\mathbf{n}_i = {}^{i}\mathbf{N}_i + {}^{i}_{i+1}\mathbf{R}\,{}^{i+1}\mathbf{n}_{i+1} + {}^{i}\mathbf{P}_{Ci} \times {}^{i}\mathbf{F}_i + {}^{i}\mathbf{P}_{i+1} \times {}^{i}_{i+1}\mathbf{R}\,{}^{i+1}\mathbf{f}_{i+1} \tag{2.20} $$

Notons que la notation ${}^{i}\mathbf{f}_i$ désigne la force exercée par le membre i-1 sur le membre i représenté dans le référentiel i.

Finalement, pour une articulation rotative, le couple articulaire est :

$$ \boldsymbol{\tau}_i = {}^{i}\mathbf{n}_i^{\,T}\,{}^{i}\hat{\mathbf{Z}}_i \tag{2.21} $$

Pour une articulation prismatique, la force articulaire est :

$$ \boldsymbol{\tau}_i = {}^{i}\mathbf{f}_i^{\,T}\,{}^{i}\hat{\mathbf{Z}}_i \tag{2.22} $$

Nous pouvons inclure la gravité dans la dynamique inverse en définissant la condition suivante :

$$ {}^{0}\dot{\mathbf{v}}_0 = -g\vec{\mathbf{v}} \tag{2.23} $$

où $\vec{\mathbf{v}}$ est un vecteur unitaire possédant la même orientation que la gravité

g est la constante gravitationnelle (scalaire)

Ceci fait en sorte que le manipulateur entier accélère dans le sens opposé de la gravité, produisant ainsi le même effet que si la gravité était présente.

2.2.2. Algorithme de Lagrange-Euler

Par la formulation de Lagrange, nous pouvons trouver les équations dynamiques d'une façon systématique. Nous devons choisir un ensemble de coordonnées généralisées

décrivant la position du système. La plupart du temps, nous utilisons directement les variables articulaires, c'est à dire les angles et longueurs décrivant la position de chacune des articulations mobiles. Le Lagrangien du système mécanique est comme suit :

$$\mathbf{L} = \mathbf{K} - \mathbf{U} \tag{2.24}$$

où $\mathbf{K} = \sum_{i=1}^{n} K_i$ est l'énergie cinétique du système, $\mathbf{U} = \sum_{i=1}^{n} U_i$ est l'énergie potentielle du système. Nous calculons l'énergie cinétique par :

$$K_i = \frac{m_i \, v_{Ci}^T \, v_{Ci} + {}^i\omega_i^T \, {}_{Ci}\mathbf{I} \, {}^i\omega_i}{2} \tag{2.25}$$

où m_i est la masse du membre

v_{Ci} est la vitesse du centre de masse du membre

L'énergie potentielle est calculée par :

$$U_i = -m_i \, {}^0\mathbf{g}^T \, {}^0\mathbf{P}_{Ci} + k_r L_r + U_{ref} \tag{2.26}$$

où ${}^0\mathbf{g}$ est le vecteur de gravité par rapport à la base, ${}^0\mathbf{P}_{Ci}$ est le vecteur de position de la masse, k_r est la rigidité du ressort monté à une articulation rotative ou prismatique (s'il est présent), L_r est la projection de la longueur du ressort sur l'axe des Z, U_{ref} est l'énergie potentielle de référence choisie pour que la valeur minimum de U soit nulle.

L'équation de Lagrange est exprimée par :

$$\frac{d}{dt}\frac{\partial \mathbf{L}}{\partial \dot{\lambda}_i} - \frac{\partial \mathbf{L}}{\partial \lambda_i} = \xi_i \tag{2.27}$$

où ξ_i sont les forces généralisées associées aux coordonnées généralisées λ_i.

Pour un manipulateur avec chaîne cinématique ouverte, un choix naturel de coordonnées généralisées est le vecteur de variables articulaires : $\lambda_i = q_i$. Dans ce cas, les forces généralisées seront les couples/forces articulaires.

La dynamique inverse peut s'écrire sous la forme Lagrange-Euler de la façon suivante :

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{J}_f(\mathbf{q})\mathbf{F}_e \tag{2.28}$$

où $\boldsymbol{\tau}$ est le vecteur de forces/couple articulaires, $\mathbf{M}(\mathbf{q})$ est la matrice des masses et inerties (définie positive et symétrique), $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ est la matrice des termes centrifuges et de Coriolis (peut être la matrice de Cristoffel, qui correspond à un choix particulier), $\mathbf{G}(\mathbf{q})$ est le vecteur des termes de gravité et $\mathbf{J}_f(\mathbf{q})$ est le Jacobien de force et $\mathbf{F}_e$, le vecteur de forces externes. Le Jacobien de force est le Jacobien de vitesse transposé.

2.3. Dynamique directe

La formulation de Lagrange-Euler peut être utilisée pour calculer la dynamique directe en réécrivant (2.28) de la façon suivante :

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})\left[\boldsymbol{\tau} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}) - \mathbf{J}_f(\mathbf{q})\mathbf{F}_e\right] \tag{2.29}$$

Pour calculer la dynamique directe avec l'algorithme de Newton-Euler, nous pouvons procéder de la manière suivante. Nous avons déjà le vecteur de couple $\boldsymbol{\tau}$ (entrée de l'algorithme). Nous pouvons obtenir les termes $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{J}_f(\mathbf{q})\mathbf{F}_e$ en calculant la dynamique inverse avec $\ddot{\mathbf{q}}$ nulle, $\mathbf{q}$ et $\dot{\mathbf{q}}$ étant connues (valeurs de l'itération précédente). Une des façons possibles de trouver $\mathbf{M}(\mathbf{q})$ consiste à évaluer cette matrice une colonne à la fois [5]. Pour ce, nous devons calculer la dynamique inverse *n* fois, où *n* est le nombre d'articulations. L'entrée pour chacun des calculs de dynamique inverse est la suivante :

- $\mathbf{q}$ du cycle précédent ;
- $\dot{\mathbf{q}} = 0$ pour toutes les articulations ;
- $\ddot{q}_i = 1$ où i est le numéro de la colonne évaluée ;
- $\ddot{q}_k = 0$ pour k = 1 à *N* sauf i ;
- $g = 0$.

Avec (2.29), nous sommes en mesure de calculer l'accélération articulaire $\ddot{\mathbf{q}}$. Pour obtenir $\dot{\mathbf{q}}$ et $\mathbf{q}$, il suffit d'intégrer deux fois $\ddot{\mathbf{q}}$ par rapport au temps.

2.4. Sommaire

Nous avons vu comment modéliser la cinématique directe d'un manipulateur robotique. En plus d'être un pré-requis pour le calcul de la dynamique inverse, la cinématique directe nous permet de calculer le Jacobien de vitesse. Le Jacobien inverse de vitesse et

sa dérivée nous permettent de former les équations de calcul des vitesses et accélérations articulaires en fonction des positions articulaires et des vitesses et accélérations gallilléennes. Ces équations serviront à transformer des trajectoires de vitesse et accélération gallilléennes en trajectoires de vitesse et accélération articulaires pour l'étude de cas du chapitre V.

Ensuite, nous avons présenté les équations de calcul de la dynamique inverse selon les formulations de Newton-Euler et Lagrange-Euler. Ces équations permettent de simuler le modèle de la dynamique directe du manipulateur en considérant la gravité, les forces centrifuges et de Coriolis de même que les forces et couples d'inertie. Ces équations peuvent aussi servir à calculer des lois de commande basées sur le modèle du robot comme par exemple un retour d'état complet, un contrôleur par modèle de référence, une anticipation, etc.

CHAPITRE III

COMMANDE ADAPTATIVE POSITION/FORCE

La commande de robots entrant en contact avec l'environnement requiert l'utilisation de contrôleurs de position et de force. Nous posons que les charges déplacées par le robot sont variables. Par exemple, nous pouvons changer l'outil du manipulateur pour effectuer une tâche précise. Pour commander le robot en position de façon efficace, il faut connaître les masses des membres du robot en tout temps. Pour ce faire, le contrôleur de position doit comprendre un identificateur de paramètres.

Nous posons que la géométrie de l'environnement du robot est connue selon une certaine précision. De plus, la rigidité des surfaces de l'environnement est variable. Lors de l'exécution d'une tâche pour laquelle un contact entre le robot et un objet est nécessaire, l'erreur de modélisation cinématique et de rigidité peuvent influencer grandement le comportement du robot. Le contrôleur de force doit être caractérisé par une certaine robustesse lui permettant de commander le manipulateur sur une plage d'opération suffisamment grande pour accomplir les tâches assignées.

Le contrôleur position /force et les trajectoires qui y sont envoyées doivent être coordonnés en fonction des tâches à effectuer par le robot. Pour ce faire, nous concevons un planificateur de tâches. Ce superviseur permet aussi aux contrôleurs de respecter certaines stratégies améliorant la commande lorsqu'il y a contact entre le robot et l'environnement.

3.1.Contrôleur adaptatif de position

Nous utilisons l'algorithme de Slotine et Li [20] pour concevoir notre contrôleur adaptatif de position. Le problème auquel répond le contrôleur est le suivant. Pour une trajectoire sans contrainte donnée, le contrôleur doit estimer les masses des membres et faire suivre la trajectoire désirée au manipulateur (annuler l'erreur de position et de vitesse). La figure 3.1 présente un schéma du contrôleur adaptatif de position.

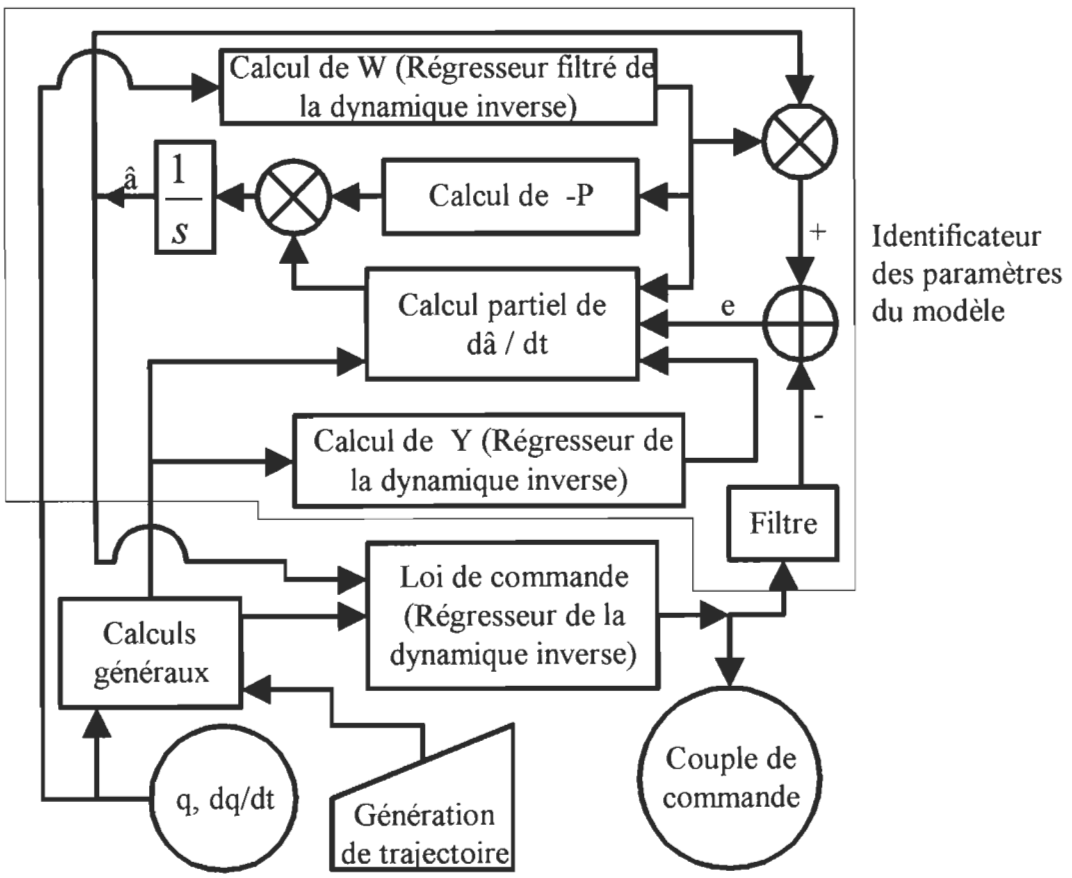

Figure 3-1 : Schéma du contrôleur adaptatif de position

La loi de commande comporte un terme d'anticipation compensant les non linéarités du système comme par exemple, la gravité. Cette loi de commande comprend aussi un retour de position et de vitesse articulaires équivalant à un contrôleur proportionnel dérivatif.

Définissons d'abord quelques variables qui seront utilisées dans les équations suivantes. Ces variables sont calculées dans le bloc « Calculs généraux ». Premièrement, pour inclure de façon implicite le contrôleur PD et pour éviter d'utiliser l'accélération dans le retour d'état, nous utilisons une version modifiée du retour d'état (pour la vitesse et l'accélération). De plus, cette substitution permet d'éliminer l'erreur de position en régime permanent.

$$\dot{\mathbf{q}}_r = \dot{\mathbf{q}}_d - \Lambda\tilde{\mathbf{q}} \tag{3.1}$$

$$\ddot{\mathbf{q}}_r = \ddot{\mathbf{q}}_d - \Lambda\dot{\tilde{\mathbf{q}}} \tag{3.2}$$

où $\dot{\mathbf{q}}_d$ et $\ddot{\mathbf{q}}_d$ sont la vitesse et l'accélération désirées, $\mathbf{\Lambda}$ est une matrice constante dont les valeurs propres sont strictement dans le demi plan droit, $\tilde{\mathbf{q}}=\mathbf{q}-\mathbf{q}_d$ et $\dot{\tilde{\mathbf{q}}}=\dot{\mathbf{q}}-\dot{\mathbf{q}}_d$ sont les erreurs de position et de vitesse respectivement. Ensuite, nous définissons l'erreur de suivi de la trajectoire par :

$$\mathbf{s} = \dot{\tilde{\mathbf{q}}}_r = \dot{\mathbf{q}} - \dot{\mathbf{q}}_r = \dot{\tilde{\mathbf{q}}} + \Lambda\tilde{\mathbf{q}} \tag{3.3}$$

Le signal de commande est calculé par

$$\boldsymbol{\tau} = \mathbf{Y}\left(\mathbf{q}, \dot{\mathbf{q}}, \dot{\mathbf{q}}_r, \ddot{\mathbf{q}}_r\right)\hat{\mathbf{a}} - \mathbf{K}_D\mathbf{s} \tag{3.4}$$

où $\mathbf{Y}(\mathbf{q}, \dot{\mathbf{q}}, \dot{\mathbf{q}}_r, \ddot{\mathbf{q}}_r)$ est la matrice régresseur du système complet ;

$\hat{\mathbf{a}}$ est le vecteur de paramètres estimés (par exemple, les masses) ;

$\mathbf{K}_D$ est une matrice multipliant l'erreur de suivi **s** (gain du contrôleur PD).

La matrice $\mathbf{Y}(\mathbf{q}, \dot{\mathbf{q}}, \dot{\mathbf{q}}_r, \ddot{\mathbf{q}}_r)$ est définie par :

$$\mathbf{M(q)}\ddot{\mathbf{q}}_r+\mathbf{C(q,\dot{q})}\dot{\mathbf{q}}_r+\mathbf{G(q)}=\mathbf{Y(q,\dot{q},\dot{q}_r,\ddot{q}_r)a} \tag{3.5}$$

La forme de $\mathbf{Y}(\mathbf{q}, \dot{\mathbf{q}}, \dot{\mathbf{q}}_r, \ddot{\mathbf{q}}_r)$ dépend donc du choix du vecteur de paramètres a. Si nous choisissons $\mathbf{K}_D=\eta\mathbf{M}$ et $\mathbf{\Lambda}=\eta\mathbf{I}$ (où $\mathbf{I}$ est la matrice identité), la loi de commande est équivalente à :

$$\begin{aligned}\boldsymbol{\tau} &= \mathbf{M(q)}\left[\ddot{\mathbf{q}}_d - 2\eta\dot{\tilde{\mathbf{q}}} - \eta^2\tilde{\mathbf{q}}\right] + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_r + \mathbf{G}(\mathbf{q}) \\ &= \mathbf{M(q)}\left[\ddot{\mathbf{q}}_r - \mathbf{\Lambda s}\right] + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_r + \mathbf{G}(\mathbf{q})\end{aligned} \tag{3.6}$$

Cette structure de commande nous assure que les erreurs de vitesse et de position convergeront vers zéro avec une constante de temps de $1/\eta$ [20]. L'équation (3.6) est implantée dans le bloc « Loi de commande » sous la forme régresseur $\mathbf{Y}(\mathbf{q}, \dot{\mathbf{q}}, \dot{\mathbf{q}}_r, \ddot{\mathbf{q}}_r - \mathbf{\Lambda s})$ comme montré à la figure 3.1.

Ensuite, voyons comment fonctionne l'adaptation des paramètres (la masse des membres). Le principe de base de l'identification des paramètres est montré à la figure 3.2. Le couple de commande filtré et le couple estimé filtré (calculé à partir du modèle inverse du manipulateur et des variables articulaires de position et de vitesse) forment

l'erreur de prédiction. Cette erreur donne de l'information sur la justesse des paramètres estimés. Donc, l'erreur de prédiction permet implicitement d'ajuster les paramètres identifiés en fonction de l'erreur sur les paramètres. De plus, l'identificateur utilise certaines autres variables pour former l'erreur de suivi permettant d'ajuster les paramètres identifiés en fonction des erreurs de position et de vitesse. L'algorithme utilise l'erreur de suivi en supposant que l'erreur de position est partiellement causée par une erreur sur les paramètres.

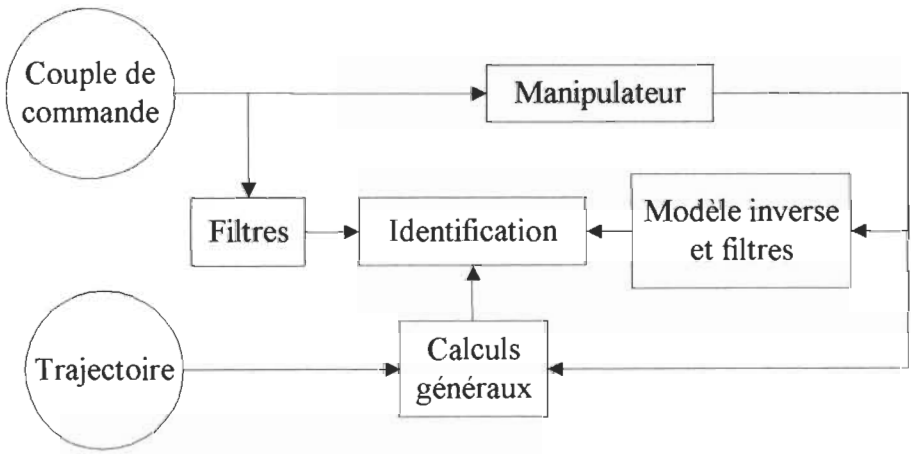

Figure 3-2 : Principe de base de l'identification des paramètres

Nous présentons ensuite l'algorithme détaillé de l'identificateur. La première étape consiste à trouver le couple observé filtré, produit par un modèle d'estimation :

$$\hat{\boldsymbol{\tau}}_f = \mathbf{W(t)}\hat{\mathbf{a}} \tag{3.7}$$

où **W(t)** est la matrice de signaux ;

" **â** " est le vecteur de paramètres estimés (les masses).

Nous devons calculer la matrice **W** sans utiliser l'accélération pour des raisons de problèmes pratiques de mesure lorsque l'accélération approche de zéro. De plus, une mesure d'accélération est relativement bruitée et nécessite l'achat d'un capteur pour chaque articulation. Pour calculer **W** sans mesure d'accélération, nous convoluons l'équation du couple de commande avec un filtre de premier ordre $w = \frac{\lambda}{s+\lambda}$ et obtenons

$$\int_0^t w(t-r)\,\boldsymbol{\tau}(r)\,dr = \int_0^t w(t-r)\left[\mathbf{M}(\mathbf{q}(r))\ddot{\mathbf{q}}(r) + \mathbf{C}(\mathbf{q}(r),\dot{\mathbf{q}}(r))\dot{\mathbf{q}}(r) + \mathbf{G}(\mathbf{q}(r))\right]dr \tag{3.8}$$

En effectuant une intégration partielle, nous obtenons le terme d'accélération filtré :

$$\begin{aligned}\int_0^t w(t-r)[\mathbf{M}\ddot{\mathbf{q}}]\,dr &= w(t-r)\,\mathbf{M}\dot{\mathbf{q}}\Big|_0^t - \int_0^t \frac{d}{dr}[w\,\mathbf{M}]\dot{\mathbf{q}}\,dr \\ &= w(0)\,\mathbf{M}(\mathbf{q})\dot{\mathbf{q}} - w(t)\,\mathbf{M}[\mathbf{q}(0)]\dot{\mathbf{q}}(0) - \int_0^t [w(t-r)\,\dot{\mathbf{M}}\dot{\mathbf{q}} - \dot{w}(t-r)\,\mathbf{M}\dot{\mathbf{q}}]\,dr\end{aligned} \tag{3.9}$$

Les termes de vitesse/Coriolis et de gravité doivent aussi être filtrés mais ils restent tel quels. Nous obtenons une équation de **W** en 3 parties ; une partie devant être filtrée (**W**00), une devant être dérivée et filtrée (**W**01) et la dernière devant être utilisée telle quelle (**W**02). Pour obtenir **W**, nous devons additionner **W**00 (filtré) et **W**01 (dérivé et filtré).

$$\mathbf{W}00 = (\mathbf{C} - \dot{\mathbf{M}})\dot{\mathbf{q}} + \mathbf{G} \tag{3.10}$$

$$\mathbf{W}01 = \mathbf{M}\dot{\mathbf{q}}$$

$$\mathbf{W}02 = \mathbf{M}\dot{\mathbf{q}}\lambda$$

$$\mathbf{W} = \mathbf{W}00\left(\frac{\lambda}{s+\lambda}\right) + \mathbf{W}01\left(\frac{\lambda s}{s+\lambda}\right) + \mathbf{W}02$$

Notons que nous calculons les matrices Jacobiennes des vecteurs **W00**, **W01** et **W02** en fonction des paramètres identifiés. Nous calculons **W** sous sa forme Jacobienne dans le bloc « Calcul de **W** ». Ensuite, dans l'algorithme, nous multiplions la matrice **W** par le vecteur de paramètres estimés (les masses) pour obtenir une estimation du couple filtré (fig 3.1).

$$\hat{\boldsymbol{\tau}}_f = \mathbf{W}\,\hat{\mathbf{a}} \tag{3.11}$$

Ensuite, nous pouvons calculer l'erreur de prédiction.

$$\mathbf{e} = \hat{\boldsymbol{\tau}}_f - \boldsymbol{\tau}_f \tag{3.12}$$

où $\boldsymbol{\tau}_f$ est le couple de commande filtré par w. Voyons maintenant comment fonctionne l'adaptation des paramètres. Nous utilisons une loi d'adaptation composée pour identifier "**a**". C'est à dire que nous adaptons "**â**" en fonction de l'erreur de prédiction "**e**" et l'erreur de suivi de la trajectoire "**s**". La loi d'adaptation est définie par

$$\dot{\hat{\mathbf{a}}}(t) = -\mathbf{P}(t)[\mathbf{Y}^T\mathbf{s} + \mathbf{W}^T\mathbf{R}(t)\mathbf{e}] \tag{3.13}$$

où **P** est la matrice de gain d'adaptation (définie positive) ; **Y** est la matrice régresseur comme définie précédemment ; **W** est la matrice de signaux ; **R** est une matrice de poids d'importance de l'information donnée par l'erreur des paramètres estimés pour la loi d'adaptation ; **s** est l'erreur de suivi de la trajectoire ; **e** est l'erreur de prédiction, c'est à dire $\left(\hat{\boldsymbol{\tau}}_f - \boldsymbol{\tau}_f\right)$. La loi d'adaptation est calculée dans les blocs « Calcul de **Y** » et « Calcul partiel de $d\hat{\mathbf{a}}/dt$ » (fig 3.1).

Pour obtenir les paramètres (les masses) « $\hat{\mathbf{a}}$ », il suffit d'intégrer « $\dot{\hat{\mathbf{a}}}$ ».

La matrice « $\mathbf{P}$ » se calcule à l'aide de

$$\dot{\mathbf{P}}=\mathbf{P}\,\rho_0\left(1-\frac{\|\mathbf{P}\|}{k_{\lim}}\right)-\mathbf{P}\mathbf{W}^{\mathrm{T}}\mathbf{W}\mathbf{P} \tag{3.14}$$

où ρ_0 est le facteur d'oubli ;

$k_{\lim}$ est la limite supérieure (positive) du module de $\mathbf{P}$.

Nous pouvons estimer le module de **P** par la trace de **P**. Le facteur d'oubli donne à l'algorithme la possibilité de réadapter les paramètres observés si les paramètres réels changent. Il est important de bien ajuster la valeur initiale de la matrice **P**. Nous devons nous assurer que le module initial de **P** ne dépasse pas la borne $k_{\lim}$. Nous calculons **–P** avec l'équation (3.14) dans le bloc « Calcul de –P » (fig 3.1).

3.2. Contrôleur non linéaire de force

La figure 3.3 montre le principe de commande en force utilisé dans ce travail (provenant de Seraji [17]). Premièrement, nous posons que le robot possède déjà son contrôleur de position et que ce dernier est toujours actif, même lorsque le robot entre en contact avec l'environnement. La position contrôlée est celle de l'effecteur (l'outil) du robot ; le vecteur de position est composé de 6 variables (3 directions et 3 angles). Nous supposons

que le contrôleur de position permet de suivre la trajectoire demandée avec un bon rendement et avec un bon découplage directionnel (une commande dans une direction n'influence pas les autres directions). Ensuite, nous implantons une boucle de commande externe pour contrôler la force exercée par le manipulateur lorsqu'il entre en contact avec l'environnement. Le principe du contrôleur de force est d'ajouter une consigne de position qui permettra d'atteindre la force désirée. Nous posons que le contrôleur de position utilise une trajectoire dans le domaine articulaire. Nous devons donc transformer la trajectoire de position galliléenne en trajectoire dans le domaine articulaire.

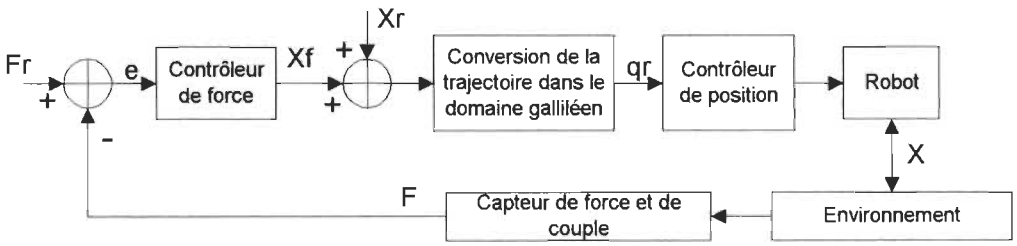

Figure 3-3 : Schéma général de commande avec contrôleur de force

Voyons maintenant comment sont modélisés le manipulateur et son environnement pour concevoir la loi de commande de force. Premièrement, nous posons que la perturbation de la position de l'effecteur Δx se comporte comme un système continu du deuxième ordre face à une commande en position Δx_c :

$$G(s) = \frac{\Delta x(s)}{\Delta x_c(s)} = \frac{k\,\Delta x_c(s) - F(s)}{ms^2 + ds + k} = \frac{c\,\Delta x_c(s) - \frac{F(s)}{m}}{s^2 + as + c} \tag{3.15}$$

où m, d et k sont respectivement la masse, l'amortissement et la rigidité de l'effecteur dans l'espace cartésien et $a=\frac{d}{m}$, $c=\frac{k}{m}$ et $F(s)$ est la force externe s'opposant au mouvement. Nous savons que le robot a un comportement non linéaire lors de mouvements de grande amplitude mais nous posons que les mouvements demandés au contrôleur de position seront suffisamment lents de sorte que la précision de ce modèle représente adéquatement le manipulateur pendant les phases d'approche et de contact avec l'environnement. L'impédance de la surface de l'environnement est modélisée par un ressort pur car le facteur dominant du modèle des surfaces rencontrées dans les tâches courantes est la rigidité (k_{rs}). Vu que nous négligeons la rigidité et la dynamique du capteur de force, la rigidité effective k_e, aussi appelée rigidité de surface est égale à k_{rs}.

Nous utilisons la loi de Hooke comme relation entre la force de contact F et le déplacement de l'effecteur :

$$F=k_e\Delta x \tag{3.16}$$

Nous substituons cette force dans (3.15) et obtenons :

$$m\Delta\ddot{x}+d\Delta\dot{x}+k\Delta x=k\Delta x_c-k_e\Delta x \tag{3.17}$$

Lorsque le manipulateur est en contact avec l'environnement, la fonction de transfert de la position de l'effecteur en fonction de la position commandée de l'effecteur est

$$\frac{\Delta x}{\Delta x_c}=\frac{k}{ms^2+ds+(k+k_e)}=\frac{c}{s^2+as+b} \tag{3.18}$$

où $b = \frac{k + k_e}{m}$.

La fonction de transfert de la force de contact en fonction de la position de perturbation demandée est :

$$\frac{F}{\Delta x_c} = \frac{ck_e}{s^2 + as + b} = k_f \frac{b}{s^2 + as + b} \tag{3.19}$$

où $k_f = \frac{ck_e}{b} = \left(\frac{1}{k} + \frac{1}{k_e}\right)^{-1}$ est la rigidité équivalente du système robot/environnement.

La figure 3.4 montre le schéma de commande employé (avec le modèle simplifié pour le manipulateur et l'environnement) :

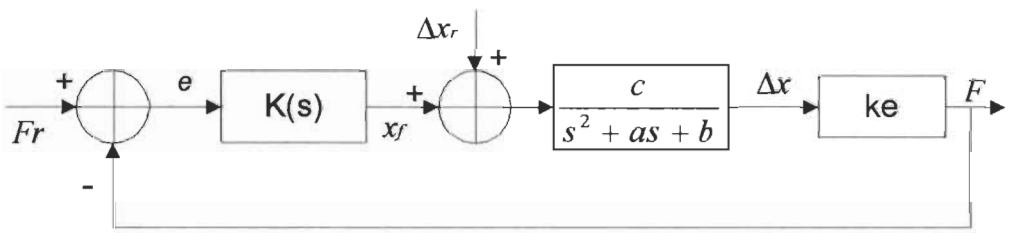

Figure 3-4 : Schéma de commande simplifié du contrôleur de force

La force de contact F varie en fonction du changement de position de l'effecteur Δx_r comme montré par

$$F(s) = \frac{k_e \bar{G}(s)}{1 + K(s) k_e \bar{G}(s)} \Delta x_r(s) + \frac{K(s) k_e \bar{G}(s)}{1 + K(s) k_e \bar{G}(s)} F_r(s) \tag{3.20}$$

où $\overline{G}(s)=\frac{c}{s^2+as+b}$ est la fonction de transfert du robot en boucle ouverte.

Nous pouvons mesurer l'efficacité du contrôleur de force par l'admittance Y définie par

$$Y(s)=\frac{v_f(s)}{e(s)} \tag{3.21}$$

où v_f est la vitesse désirée de l'effecteur et $e=F_r-F$ est l'erreur de force. Une admittance élevée implique une réaction rapide à une erreur de force, ce qui correspond à ce que nous voulons. De plus, nous savons, par la figure 3.3, que la fonction de transfert générale du contrôleur est

$$K(s)=\frac{x_f(s)}{e(s)} \tag{3.22}$$

En combinant (3.21) et (3.22), nous obtenons

$$K(s)=\frac{1}{s}Y(s) \tag{3.23}$$

Nous voyons qu'en boucle fermée, l'admittance $Y(s)$ est en fait la fonction de transfert de la vitesse de l'effecteur v_f en fonction de l'erreur de force $e(s)$. En d'autres mots, l'admittance modélise le comportement du contrôleur de force en terme de vitesse de l'effecteur face à une erreur de force changeante. Nous cherchons à obtenir une admittance $Y(s)$ de premier ordre.

$$Y(s)=k_p s+k_i \tag{3.24}$$

La fonction de transfert générale du contrôleur est donc :

$$K(s) = \frac{1}{s} Y(s) = k_p + \frac{k_i}{s} \tag{3.25}$$

La loi de commande est donc :

$$x_f(t) = k_p e(t) + k_i \int_0^t e(t) dt \tag{3.26}$$

où k_p et k_i sont les gains proportionnel et intégral.

L'avantage du modèle d'admittance de premier ordre face à l'admittance de deuxième ordre est que nous n'avons pas besoin du taux de variation de l'erreur de force $\dot{e}(t)$ pour la commande. Par contre, nous avons moins de contrôle sur le comportement des régimes transitoires et nous pouvons seulement assurer une convergence asymptotique. Pour obtenir une performance robuste face aux incertitudes et aux variations de rigidité de la surface de contact, nous introduisons un gain intégral non linéaire et fonction de la force de contact.

$$x_f(t) = k_p\, e(t) + \int_0^t \left(k_i\ \left(e(t)\right) e(t)\right) dt \tag{3.27}$$

Dans [17], le critère de stabilité de Popov est utilisé pour établir les conditions sur k_i permettant d'assurer la stabilité du système en boucle fermée. Après démonstration, nous obtenons que le gain k_i doit demeurer dans la région :

$$0 \le k_i \le a\left(k_f^{-1} + k_p\right) \tag{3.28}$$

Parmi toutes les options possibles de fonctions non linéaires respectant le critère (3.28), nous choisissons la fonction sigmoïde suivante :

$$k_i = k_0 + \frac{k_1}{1+\exp\left(-k_2 e_s\right)} \tag{3.29}$$

où k_0, k_1 et k_2 sont des constantes positives, $e_s = signe\left(F_r - F_s\right)e$ est l'erreur de force signée par rapport à la force de contact F_s qui était appliquée avant le régime transitoire permettant d'atteindre la nouvelle force de consigne F_r. La fonction "signe" est présente pour s'assurer que le gain intégral varie de la même façon indépendamment de la direction de variation de F_r par rapport à F_s. De plus, pour assurer la stabilité du système en boucle fermée, les constantes k_0 et k_1 doivent satisfaire :

$$0 < k_0 \quad \text{et} \quad \left(k_0 + k_1\right) \le a\left(k_f^{-1} + k_p\right) \tag{3.30}$$

Finalement, voici comment se déroule un contact avec cet algorithme. Premièrement, au début du contact, l'erreur de force est grande ; k_i est donc grand aussi. Ceci assure une réponse initiale rapide face à l'erreur de force. Ensuite, plus l'erreur de force converge vers zéro, plus la constante k_i est diminuée, diminuant ainsi le temps de stabilisation du contrôleur.

La sortie du contrôleur est une position cartésienne à ajouter à la trajectoire cartésienne générale du manipulateur. Mais nous avons aussi besoin de la vitesse et de l'accélération

pour utiliser l'algorithme de commande en position de Slotine&Li [20]. Par conséquent, nous proposons une méthode permettant d'obtenir la position, vitesse et accélération approximées à partir de la position de sortie du contrôleur de force. Pour trouver la vitesse et l'accélération, nous dérivons respectivement une et deux fois. Pour éviter de grandes valeurs de vitesse et d'accélération, nous appliquons un filtre du deuxième ordre sur la position, vitesse et accélération (nous appliquons le même filtre sur les trois pour qu'elles demeurent consistantes entre elles). Les fonctions de transfert permettant de calculer les position, vitesse et accélération approximées en fonction de la position provenant du contrôleur PI sont respectivement :

$$\begin{aligned} p_a &= \frac{10000}{s^2 + 200s + 10000} \\ v_a &= \frac{10000s}{s^2 + 200s + 10000} \\ a_a &= \frac{10000s^2}{s^2 + 200s + 10000} \end{aligned} \tag{3.31}$$

La figure 3.5 montre l'emplacement des filtres convertissant la trajectoire de position calculée par le contrôleur de force en trajectoires de position, vitesse et accélération. Le bloc « mux » sert à concaténer les trajectoires de position, vitesse et accélération en un seul vecteur.

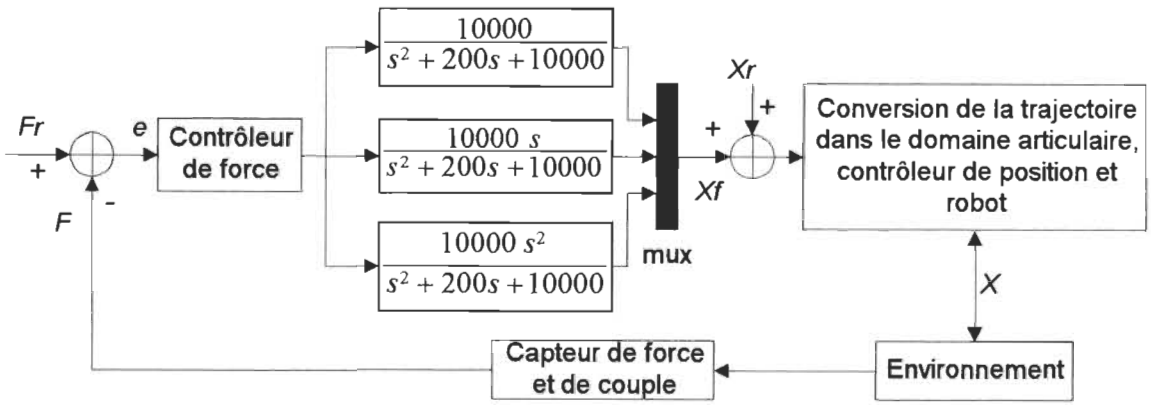

Figure 3-5 : Emplacement des filtres de la trajectoire de position fournie par le contrôleur de force

3.3. Planification des tâches et stratégies d'approche

La figure 3.6 montre l'emplacement et le rôle du planificateur de tâches à l'intérieur du schéma de commande. Les contrôleurs de position et de force doivent être coordonnés en fonction des tâches à exécuter par le robot. Nous utilisons un planificateur de tâches pour superviser les contrôleurs. Ce planificateur gère aussi les générateurs de trajectoires. Il désactive l'identification des masses du contrôleur adaptatif de position avec le signal « Scp » lorsqu'il y a contact entre l'effecteur et l'environnement. Cette désactivation est nécessaire pour éviter que les paramètres observés divergent. En effet, les contraintes physiques imposées par un contact peuvent causer l'identification de très grandes masses et mener à une commande en position de mauvaise qualité. Notons que la trajectoire « Xf » comprend la position, la vitesse et l'accélération galliléennes comme défini en (3.31). Les signaux « Xr », « Xt » et « qt » comprennent aussi des positions, vitesses et

accélérations. Le signal « Xr » contient la trajectoire de consigne. Le signal « Xt » comprend les trajectoires totales dans le domaine galliléen tandis que le signal « qt » contient les trajectoires totales dans le domaine articulaire.

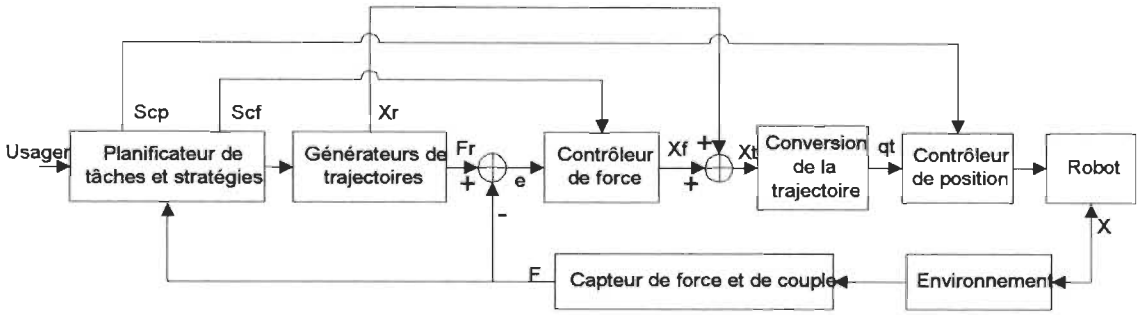

Figure 3-6 : Principe de fonctionnement du planificateur de tâches

De façon plus précise, le planificateur supervise les tâches suivantes. Il permet l'identification des masses en commandant une trajectoire aller retour en position. Ensuite il peut commander la saisie d'un objet. Il existe différentes stratégies pouvant être appliquées pour améliorer les performances du système pendant l'exécution de ces tâches. Pour approcher et saisir un objet, il génère une trajectoire de position pour amener l'effecteur près de l'objet. Le planificateur connaît approximativement la position de l'objet, mais il peut y avoir de grandes forces si l'effecteur dépasse le point de contact. La saisie s'effectue donc en deux étapes : la saisie à force faible et l'application de la force de consigne. Le planificateur active donc le contrôleur de force et commande l'effecteur selon deux trajectoires continues comme montré à la figure 3-7. Lors de la relâche du contact, le planificateur doit aussi appliquer deux trajectoires de force successives : une trajectoire diminuant la force de contact à un faible pourcentage de la consigne et une trajectoire permettant aux doigts de quitter l'objet (force de consigne

nulle). Il est désavantageux de tenter de ramener la consigne de force à zéro en une seule étape car il pourrait se produire des rebondissements. Lors de la relâche complète du contact, le planificateur désactive le contrôleur de force. Il faut aussi remettre l'intégrateur du contrôleur de force à zéro avec le signal « Scf » pour éviter qu'il y ait un biais permanent sur la trajectoire de position.

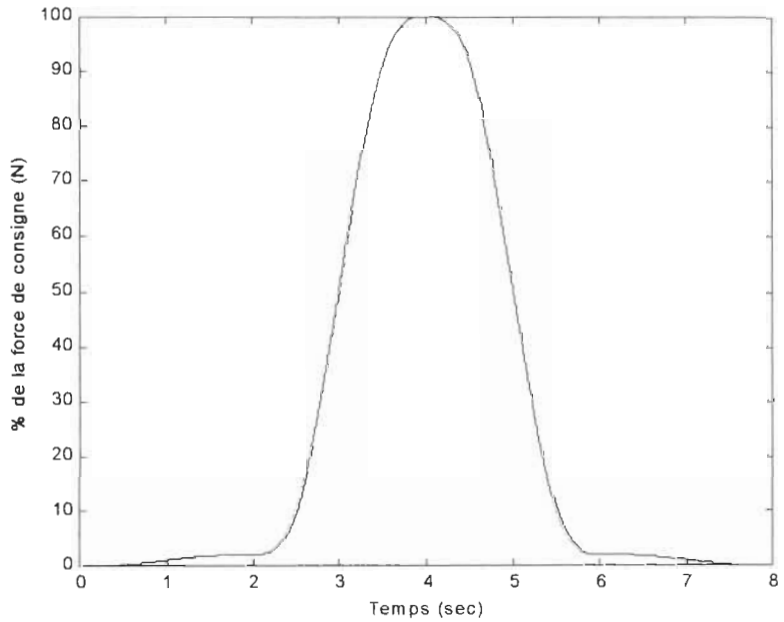

Figure 3-7 : Trajectoire de force typique utilisée pour la saisie

Pour déplacer le manipulateur dans l'espace non contraint en force, nous avons produit un générateur de trajectoire. Nous avons calculé une trajectoire continue pour l'accélération (premier ordre), la vitesse (deuxième ordre) et la position (troisième ordre). Le principe de fonctionnement est le suivant. En spécifiant la durée désirée de la trajectoire, la position initiale et finale, l'algorithme calcule automatiquement la position, vitesse et accélération en fonction du temps (en temps réel). Avec ce générateur de trajectoire, nous atteignons une vitesse et accélération maximales fonctions de la distance

à parcourir. La trajectoire est divisée en 4 parties de durées égales, la deuxième et troisième parties étant confondues. Les calculs d'accélération sont:

$$\begin{aligned} a &= D\,t && \text{pour } 0 \le t \le {t_f}/{4} \\ a &= -D\,t + \frac{D\,t_f}{2} && \text{pour } {t_f}/{4} \le t \le {3t_f}/{4} \\ a &= D\,t - D\,t_f && \text{pour } {3t_f}/{4} \le t \le t_f \end{aligned} \tag{3.32}$$

où D est la dérivée de l'accélération (cette variable est calculée en fonction de t_f); t est le temps relatif au début de la trajectoire; t_f est le temps final (durée) de la trajectoire.

Les calculs de vitesse sont :

$$\begin{aligned} v &= \frac{D\,t^2}{2} && \text{pour } 0 \le t \le {t_f}/{4} \\ v &= \frac{-D}{2}\left(t^2 - t_f\,t + \frac{{t_f}^2}{8}\right) && \text{pour } {t_f}/{4} \le t \le {3t_f}/{4} \\ v &= D\left(\frac{t^2}{2} - t_f\,t + \frac{{t_f}^2}{2}\right) && \text{pour } {3t_f}/{4} \le t \le t_f \end{aligned} \tag{3.33}$$

Les calculs de position sont :

$$p = \frac{D\,t^3}{6} + p_0 \qquad \text{pour } 0 \le t \le {t_f}/{4}$$

$$p = \frac{D}{2}\left(\frac{-t^3}{3} + \frac{t_f\, t^2}{2} - \frac{t_f^{\,2}\, t}{8} - \frac{t_f^{\,3}}{48}\right) + \frac{p_0 + p_f}{2} \qquad \text{pour } t_f/4 \le t \le 3t_f/4 \tag{3.34}$$

$$p = D\left(\frac{t^3}{6} - \frac{t_f\, t^2}{2} + \frac{t_f^{\,2}\, t}{2} - \frac{tf^3}{6}\right) + p_f \qquad \text{pour } 3t_f/4 \le t \le t_f$$

Pour trouver la constante D, nous devons comparer la première et la deuxième formule de position pour $t = t_f/4$. Nous obtenons :

$$\frac{D\, t_f^{\,3}}{384} + p_0 = \frac{-D\, t_f^{\,3}}{384} + \frac{D\, t_f^{\,3}}{64} - \frac{D\, t_f^{\,3}}{64} - \frac{D\, t_f^{\,3}}{96} + \frac{p_f + p_0}{2} \tag{3.35}$$

Après calculs et réarrangements, nous obtenons :

$$D = \frac{32\left(p_f - p_0\right)}{t_f^{\,3}} \tag{3.36}$$

où p_f est la position finale de la trajectoire ; p_0 est la position initiale de la trajectoire.

3.4. Sommaire

Dans ce chapitre, nous avons présenté les principaux éléments de lois de commande position/force. Ces lois de commande seront utilisées comme exemple pour l'étude de cas du chapitre V.

Ensuite, nous avons décrit le planificateur de tâches. Cet élément permettra aux contrôleurs de fonctionner efficacement face aux diverses situations qui seront étudiées au chapitre V.

Finalement, nous avons vu comment les équations du générateur de trajectoire de position et de force nous permettent d'obtenir une trajectoire continue pour les position, vitesse et accélération. Ce générateur, piloté par le planificateur de tâches, servira à produire les trajectoires de position et de force dans l'étude de cas du chapitre V.

CHAPITRE 4

GÉNÉRATEUR AUTOMATIQUE DE CODE ET PROGRAMMES UTILITAIRES

Le temps consacré à la modélisation et à l'écriture du code de simulation des robots et de ses contrôleurs est un problème majeur. Nous constituons un générateur de code produisant automatiquement les modèles des robots et de son contrôleur de position à partir d'une brève description du système. La description requise comprend les paramètres de la convention Denavit-Hartenberg modifiés (décrivant la géométrie du modèle), les paramètres dynamiques (poids, position et forme des masses) et les paramètres du contrôleur de position. Un des avantages de la génération automatique est la simplification des calculs par rapport aux particularités du système modélisé. Il s'en suit une amélioration du temps d'exécution de la simulation. Le générateur de code utilise l'algorithme de Newton-Euler pour calculer la dynamique inverse et il peut produire les codes selon la formulation de Newton-Euler ou Lagrange-Euler. Les programmes constituant le générateur de code sont présentés en annexe A. Nous utilisons les programmes présentés dans [27] et [28] pour obtenir les codes de calcul de la cinématique et de la dynamique sous la forme de Lagrange.

Pour entrer les paramètres numériques du robot et des contrôleurs, nous avons conçu un menu utilisant l'interface graphique de Matlab®. Le générateur de code est ensuite appelé par ce menu. Nous décrivons aussi dans ce chapitre les blocs de calcul matriciel et robotique ayant été utilisés pour constituer la version du modèle entièrement écrite en Simulink™.

4.1. Programmes d'initialisation

Nous avons constitué un programme *Matlab*® servant à initialiser les données de la simulation du manipulateur. Pour ce faire, nous avons utilisé l'interface visuel de *Matlab*® (GUI) par le biais du programme de conception d'interface « GUIDE ». Les paramètres numériques à initialiser sont les suivants :

- Nombre d'articulations ;
- Gravité ;
- Paramètres D-H modifiés ;
- Masses ;
- Position des centres de gravité des masses sur les membres ;
- Tenseurs d'inertie des masses ;
- Types d'articulation (prismatique ou rotative) ;
- Paramètres des contrôleurs de position et de force.

L'interface conçu est représenté à la figure 4.1. Les programmes *Matlab*® composant l'interface sont présentés à l'annexe B.

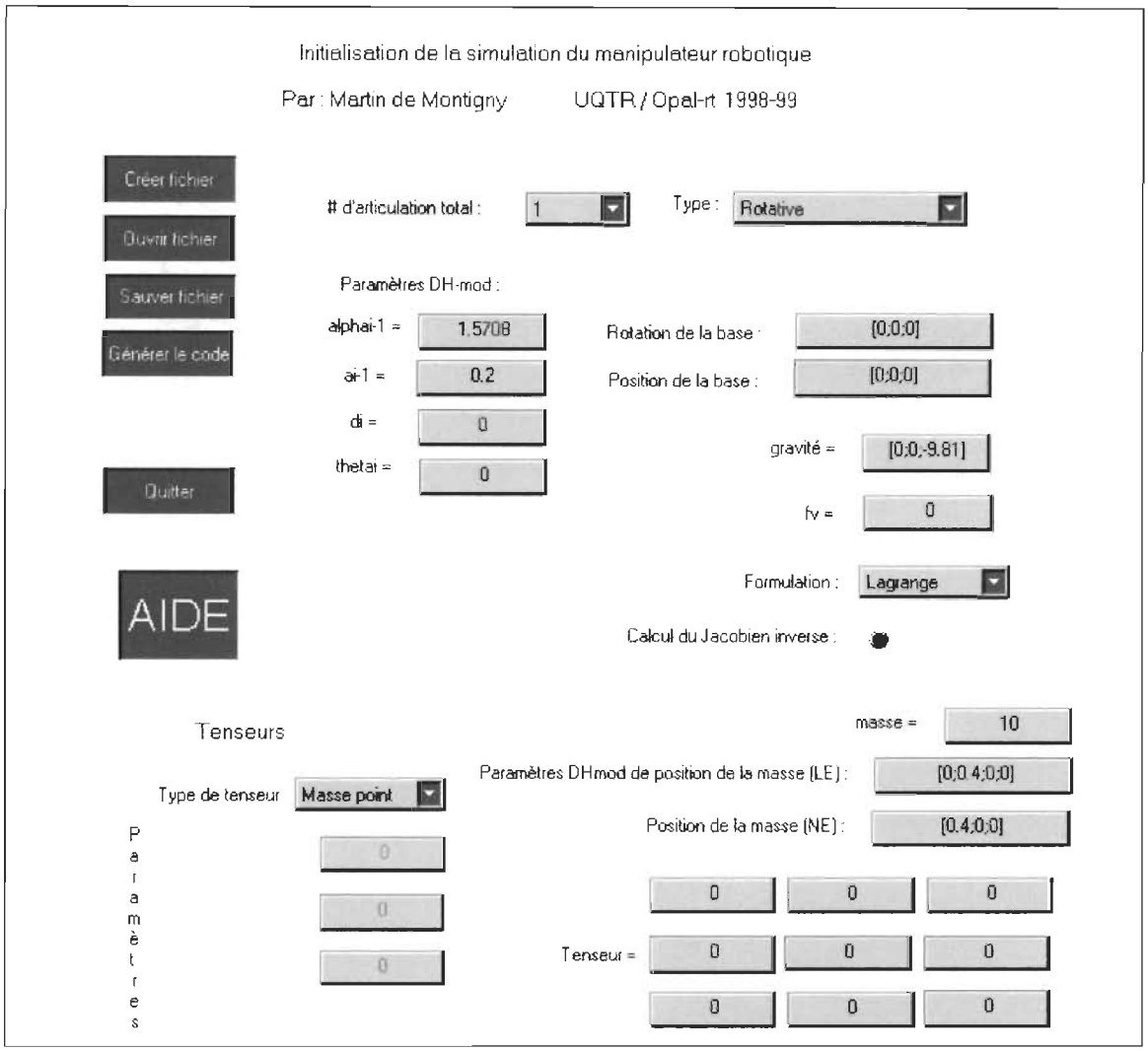

Figure 4-1 : Interface de l'utilisateur

4.2. Générateur automatique de code pour l'algorithme de Newton-Euler

Nous pouvons voir à la figure 4-2 un diagramme simplifié montrant le fonctionnement général de la génération de code automatique sous la forme de Newton-Euler.

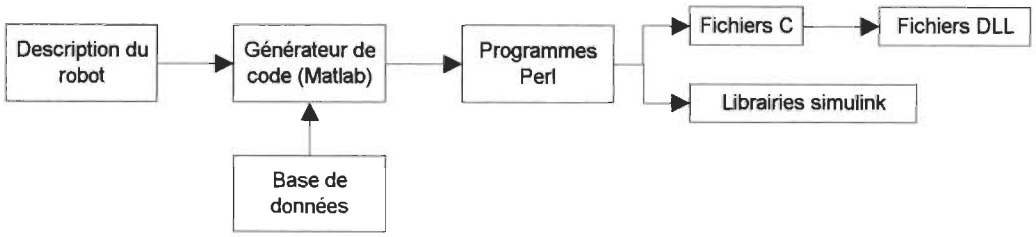

Figure 4-2 : Principe de fonctionnement du générateur de code automatique

Le moteur de la génération de code est différent dépendamment de la formulation utilisée pour représenter le modèle dynamique du robot. Par contre, le principe est le même. Tout d'abord, il faut entrer les paramètres du robot dans l'interface graphique prévu à cet effet. Ensuite, pour générer automatiquement les codes sources, il suffit d'activer le générateur de code par un bouton de l'interface. Le générateur de code (écrit en code *Matlab*®, version 5.3) calcule les équations de la dynamique inverse selon la méthode de Newton-Euler. Ensuite, les équations sont transformées selon le choix de formulation effectué par l'utilisateur dans l'interface graphique. Les équations résultantes sont écrites dans des fichiers texte par des instructions *Matlab*® (à l'intérieur du générateur de code) et des programmes *Perl*. La compilation des codes source est effectuée automatiquement à l'aide de *Watcom C++ 11.0*. Les fichiers résultant de la compilation sont des fichiers DLL qui doivent être appelés de *Simulink*™ par une « *s-function* ». Les blocs d'appel des fonctions compilées sont générés sous forme de librairies pour le calcul de la cinématique directe et de la dynamique inverse du modèle utilisant la formulation de Newton-Euler. La figure 4-3 montre un diagramme de la hiérarchie des programmes de génération de code automatique par l'algorithme de Newton-Euler.

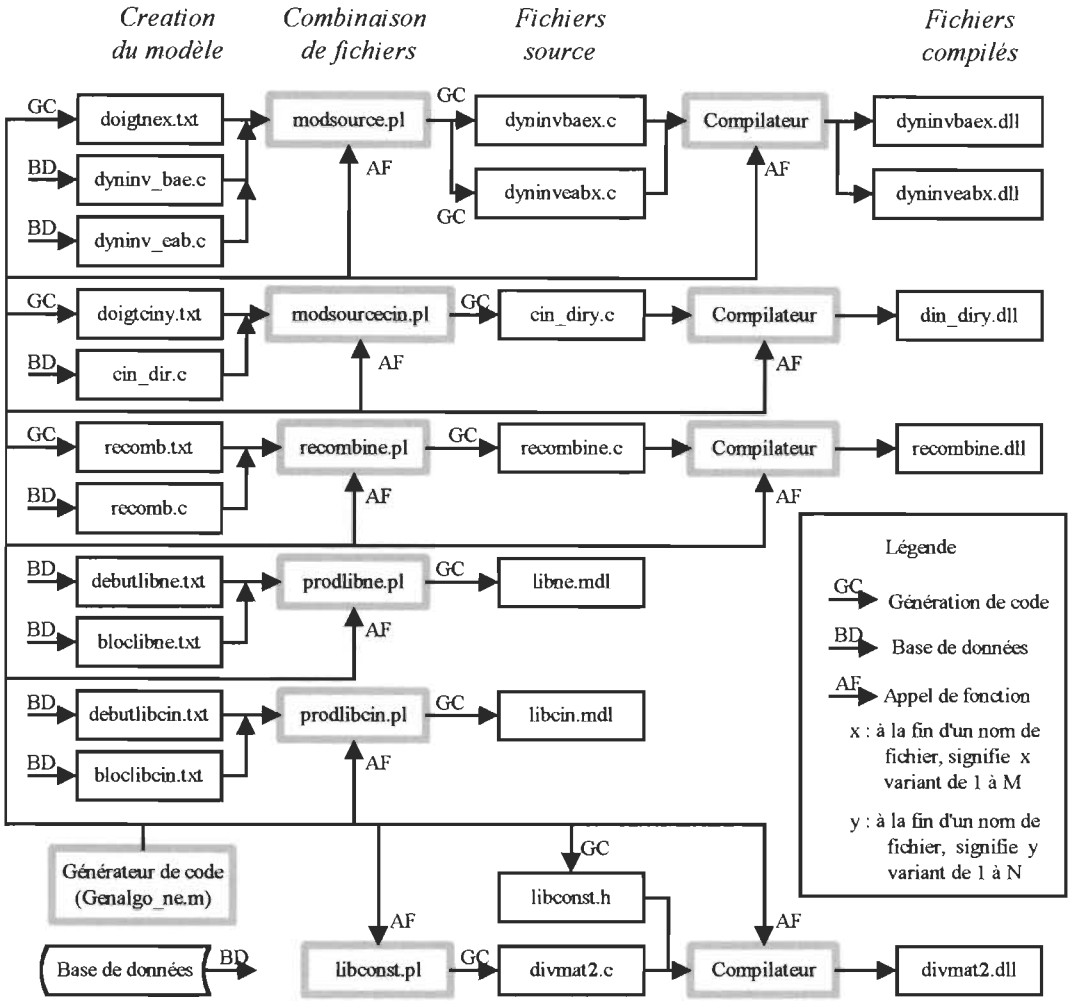

Figure 4-3 : Structure hiérarchique des programmes impliqués dans la génération automatique de code avec l'algorithme de Newton-Euler

Décrivons maintenant les programmes utilisés et créés lors de la génération de code sous la forme de Newton-Euler. Nous pouvons voir les programmes nécessaires à la génération du code en annexe A. Le programme *Matlab*® « Genalgo_ne.m » est le noyau de la génération de code automatique. C'est lui qui appelle tous les autres codes servant à générer les fichiers sources de cinématique et de dynamique. Premièrement, il calcule les équations cinématiques et dynamiques en fonction de la topologie et des masses décrites dans l'interface de l'utilisateur. Ensuite, il écrit ces équations dans des fichiers

texte « doigtciny.txt » (où y varie de 1 à N, le nombre total d'articulations) et « doigtnex.txt » (où x varie de 1 à M, le nombre d'articulations actives). La coordonnée x désigne le numéro de l'articulation pour laquelle le couple est calculé, que ce soit pour calculer un élément de la matrice des masses ou du vecteur des forces centrifuges, des termes de Coriolis et de gravité. Le fichier « recomb.txt » contient les équations de mise en ordre pour former le programme de recombinaison « recombine.c ». Ce dernier sert à réorganiser les données (cinématique directe, vitesse et accélération articulaires, données relatives aux masses etc.) d'un ordre fonctionnel à un ordre articulaire. C'est à dire que nous avons en entrée, dans l'ordre, les vitesses et accélérations articulaires, les masses (amplitude, position et forme) et la cinématique directe. Nous avons en sortie la vitesse et accélération de la première articulation, la masse du premier membre (de même que sa position et sa forme) et la cinématique directe de la première articulation. Ensuite, nous avons ces mêmes paramètres pour la deuxième articulation et ainsi de suite pour les suivantes. Les blocs d'appel des routines produites par le générateur de code sont présentées à l'annexe C. Le fichier « libconst.h » contient trois constantes nécessaires à la compilation du fichier source « divmat2.c ». Ces constantes sont M, M^2 et $M + M^2$ où M est le nombre d'articulations actives.

La base de données est composée de fichiers texte. Ces fichiers sont les coquilles des codes source C et des librairies produits par le générateur de code. Ces fichiers requièrent en fait seulement quelques lignes de code pour devenir les fichiers source finaux. Les fichiers « dyninv_bae.c » et « dyninv_eab.c » sont les coquilles des fichiers source de dynamique inverse articulaire (calculant respectivement l'itération de la base à

l'effecteur et de l'effecteur à la base). Le fichier « cin_dir.c » est la coquille nécessaire à la génération des codes de calcul de la cinématique directe articulaire. Le fichier « recomb.c » est la coquille de la fonction de recombinaison. Nous décrivons maintenant la procédure de constitution des deux librairies *Simulink*™ de blocs d'appel de calcul de cinématique directe et de dynamique inverse articulaires. Les blocs sont générés pour *Simulink*™ version 3. Les fichiers « debutlibcin.txt » et « debutlibne.txt » sont respectivement les coquilles composant le début des librairies de cinématique directe et de dynamique inverse. Le fichier « bloclibcin.txt » est ajouté au fichier *Simulink*™ de cinématique directe autant de fois qu'il y a d'articulations. Le fichier « bloclibne.txt » est quant à lui ajouté au fichier de dynamique inverse autant de fois qu'il y a d'articulations actives.

Le transfert des équations des fichiers texte aux fichiers source s'effectue à l'aide de programmes *perl*. Ces programmes sont respectivement « modsource.pl », « modsourcecin.pl » et « recombine.pl » pour la dynamique inverse, la cinématique directe et la fonction de recombinaison. Les programmes *perl* « prodlibne.pl » et « prodlibcin.pl » permettent de construire les librairies *Simulink*™ contenant les blocs d'appel de la dynamique inverse « libne.mdl » et de la cinématique directe « libcin.mdl ». Le programme « libconst.pl » créé le fichier « libconst.h ».

Les codes source produits sont « dyninvbaex.c » (x varie de 1 à M) et « dyninveabx.c » pour le calcul de la dynamique inverse (respectivement l'itération de la base à l'effecteur et l'itération de l'effecteur à la base). Les programmes « cindiry.c » (y varie de 1 à N)

servent à calculer les équations de la cinématique directe. Le code « recombine.c » est la fonction de recombinaison décrite précédemment. Finalement, le générateur compile l'algorithme de résolution d'équation par décomposition LU [29]. Cet algorithme permet de calculer l'accélération articulaire en fonction de la matrice des masses/inerties et du vecteur des termes de Coriolis et de gravité. L'algorithme par décomposition LU profite du fait que la matrice des masses/inerties est symétrique. L'ordre des matrices et vecteurs d'entrée et de sortie est automatiquement ajusté à l'aide du fichier « libconst.h » (produit par « genalgo_ne.m »).

4.3. Générateur automatique de code pour l'algorithme de Lagrange-Euler

La génération de code sous la forme de Lagrange-Euler est similaire à la génération de code utilisant la formulation de Newton-Euler décrite à la section 4.2. Les équations de la cinématique directe et de la dynamique inverse sont calculées par les logiciels *cindir* [27] et *dynam* [28]. Les programmes *Cindir* et *dynam* modifiés pour le générateur de code sont montrés à l'annexe D. Il y a deux différences majeures : tout d'abord, pour transformer les équations de la formulation Newton-Euler à Lagrange-Euler, le programme *dynam* effectue plusieurs opérations à partir de l'expression du vecteur de forces/couples articulaires. Premièrement, pour calculer les éléments de coordonnées (i, j) de la matrice des masses, le générateur de code dérive l'élément « i » du couple articulaire par rapport à l'accélération de l'articulation « j ». Ensuite, pour calculer le vecteur des termes de gravité, le générateur de code remplace les vitesses et accélérations articulaires par zéro. Finalement, pour trouver le vecteur des termes de produits de

vitesse et de Coriolis, le générateur de code soustrait la matrice des masses (multipliée par les accélérations articulaires) et le vecteur des termes de gravité au vecteur de forces/couples articulaires.

La seconde différence majeure se situe au niveau de la structure des blocs *Simulink*™ d'appel des fonctions générées. Le modèle sous la forme de Lagrange-Euler utilise une coquille *Simulink*™ fixe; seulement les codes C changent d'un modèle de robot à l'autre.

Nous pouvons voir un résumé du cheminement du générateur de code avec formulation de Lagrange-Euler à la figure 4.4.

La structure en boucle fermée des équations sous la forme de Lagrange-Euler rend possible certaines simplifications. Premièrement, les opérations de puissance sont remplacées par des opérations de multiplications successives car, pour un petit exposant entier, la multiplication est plus efficace que la puissance. Ensuite, le générateur de code analyse les équations et fait la liste de toutes les opérations trigonométriques utilisées (sinus et cosinus). Il transforme ensuite les opérations trigonométriques en entrées ; les opérations trigonométriques sont effectuées à l'extérieur des modèles et les résultats sont distribués à tous les blocs du modèle et du contrôleur de position. Finalement, le générateur de code analyse les équations pour trouver toutes les multiplications entre constantes. Il effectue ensuite un pré calcul de ces multiplications et remplace les constantes et leurs opérateurs de multiplication par des constantes uniques.

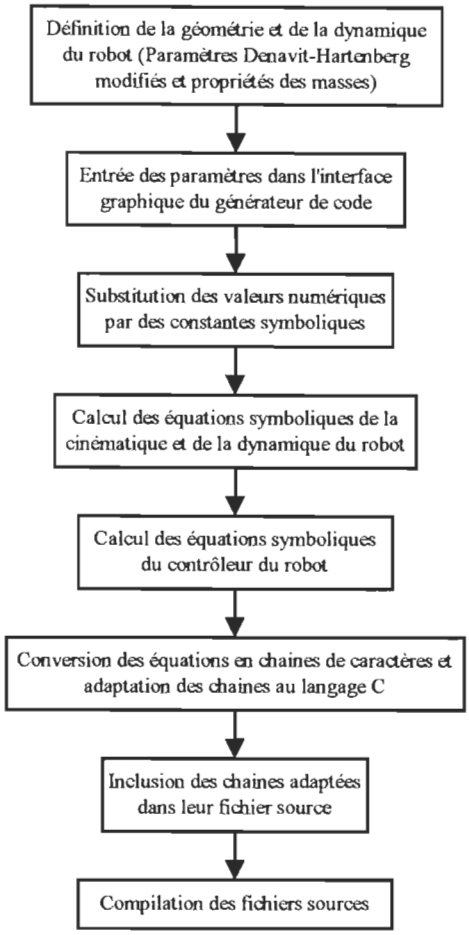

Figure 4-4 : Fonctionnement de la génération de code avec l'algorithme de Lagrange-Euler

Nous décrivons ensuite, à l'aide de la figure 4-5, la procédure exacte entreprise lors de la génération de code utilisant la formulation de Lagrange-Euler.

4.4. Génération de lois de commande à partir de modèles

Nous désirons concevoir des lois de commande permettant de contrôler la position de l'effecteur du manipulateur modélisé à l'aide des outils décrits précédemment. Pour ce faire, nous pouvons utiliser le modèle sous forme régresseur. En effet, la matrice

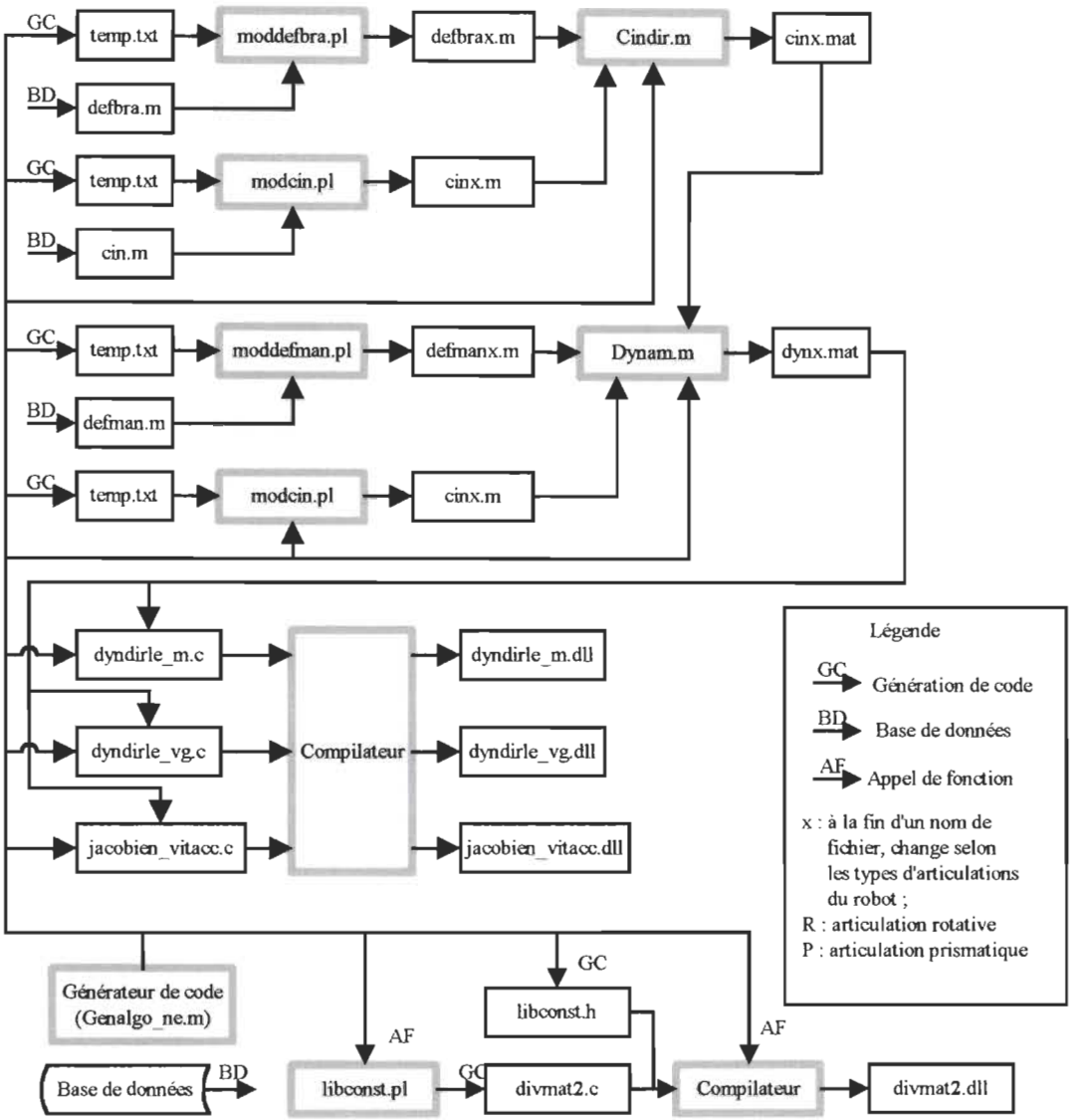

Figure 4-5 : Schéma d'interaction des fichiers nécessaires au générateur de code utilisant la formulation de Lagrange-Euler

régresseur du modèle du robot nous permet d'obtenir, par exemple, un contrôleur d'anticipation, un contrôleur par retour d'état ou certaines matrices régresseur modifiées (comme la matrice **W** du contrôleur de Slotine et Li [20]).

Étant donné que ces contrôleurs peuvent être calculés directement à partir de la matrice régresseur du robot, nous avons inclus la génération automatique du contrôleur de Slotine

et Li [20] dans l'outil de modélisation. Nous pourrions aussi facilement générer la matrice des masses, le vecteur des termes de force centrifuge ou le vecteur des termes de gravité. Les fichiers générés automatiquement sont :

calcul_w.c : Calcul de la matrice **W** en 2 parties (partie à filtrer, partie à dériver et filtrer);

calcul_pp.c : Calcul de la matrice $\dot{\mathbf{P}}$;

calcul_y_a.c : Calcul partiel de $\dot{\hat{\mathbf{a}}}$;

loi_commande.c : Calcul de l'anticipation (modèle inverse du robot).

4.5. Librairie de calcul matriciel et robotique

Nous avons conçu pour nos besoins une librairie de fonctions robotiques et de calcul matriciel. Cette librairie est représentée à la figure 4-6.

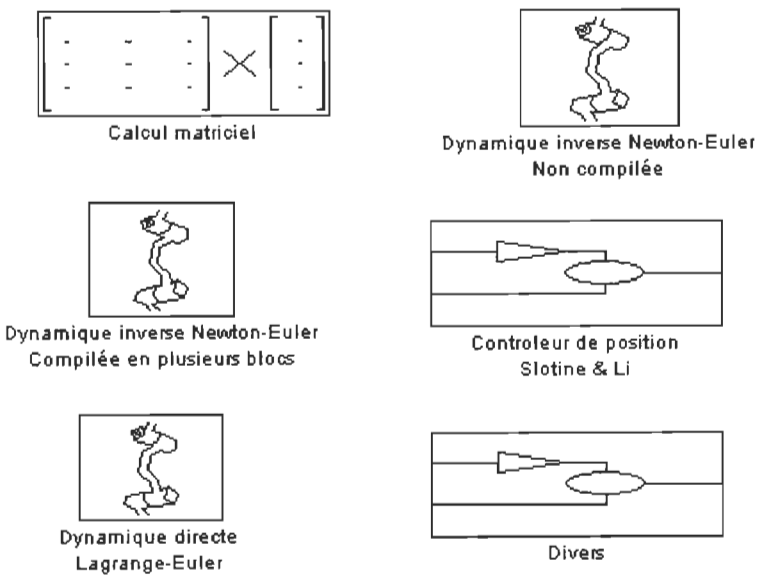

Figure 4-6 : Librairie de calcul matriciel et robotique

Les blocs de calcul sont écrits sous forme de « *s-function* » (en langage C). Notons que nous n'utilisons pas la plupart de ces blocs (ex : les blocs non compilés ou la dynamique inverse compilée en plusieurs blocs) dans la version actuelle du modèle car nous avons évolué vers un algorithme de plus gros grains (ayant des fonctions C plus grosses et moins nombreuses). Les codes source de ces fonctions ainsi que leurs blocs d'appel Simulink™ sont présentés à l'annexe C. Le groupe « Calcul matriciel » contient des fonctions de produit matrice-vecteur, matrice-matrice, produit vectoriel, transposée, résolution d'équation etc. Le groupe « Dynamique inverse Newton-Euler non compilée » contient des blocs permettant de calculer la plupart des équations de Newton-Euler en utilisant uniquement des blocs *Simulink*™. Le groupe « Dynamique inverse Newton-Euler compilés en plusieurs blocs » contient des blocs compilés (langage C) effectuant les mêmes calculs que les blocs non compilés. Le groupe « Controleur de position Slotine & Li » contient les blocs d'appel permettant de calculer l'algorithme de Slotine et Li ; les blocs *Simulink*™ demeurent toujours les mêmes, ce sont seulement les codes qui changent. Le groupe « Dynamique directe Lagrange-Euler » contient le bloc de calcul de la dynamique directe utilisant la formulation de Lagrange-Euler. Comme pour l'algorithme de Slotine et Li, la structure de ce bloc (la coquille du code) reste constante tandis que le code change en fonction du robot modélisé. Finalement, le groupe « Divers » contient le bloc d'appel de la fonction de recombinaison, un bloc de calcul des vitesses et accélérations articulaires en fonction des positions articulaires, vitesses et accélérations cartésiennes en utilisant le Jacobien. Il contient aussi une ancienne version (plus générale) du calcul de la dynamique inverse selon la formulation de Newton-Euler.

4.6. Sommaire

Dans ce chapitre, nous avons présenté l'interface de l'utilisateur et le fonctionnement du générateur de code. L'interface permettra d'entrer les paramètres numériques décrivant un manipulateur conçu pour l'étude de cas du chapitre V. Ensuite, le générateur de code nous permettra d'obtenir automatiquement les codes de simulation du robot décrit dans l'interface.

Ensuite, nous avons présenté une librairie de calcul matriciel et robotique. Cette librairie contient des blocs d'appel des fonctions générées automatiquement pour l'étude de cas du chapitre V et des fonctions de calcul matriciel et robotique utilisées dans des versions précédentes du modèle utilisant la formulation de Newton-Euler. Ces versions précédentes de modèles de robot peuvent servir dans le domaine de l'éducation car, étant constituées surtout de blocs Simulink, il est aisé de visualiser les équations des modèles.

CHAPITRE V

MODÉLISATION ET SIMULATION D'UNE MAIN ROBOTIQUE EFFECTUANT UNE MANIPULATION

L'étude de cas servant à valider le générateur de code consiste en la modélisation, la commande et la simulation d'une main robotique en interaction avec l'environnement. Les doigts de la main sont indépendants et sont modélisés à l'aide du générateur de code automatique. L'étude de cas comprend plusieurs volets. Premièrement, nous présentons une description du manipulateur et de son environnement. Nous montrons la structure géométrique et les propriétés dynamiques des doigts et de l'objet saisi. Nous utilisons le principe de commande expliqué au chapitre 4. Les programmes servant à visualiser les résultats de simulation sont ensuite présentés. Ensuite, nous tentons d'optimiser les résultats de commande force/position et d'améliorer la robustesse des algorithmes face à la variation des paramètres de l'environnement et du robot. L'algorithme de Lagrange-Euler est comparé à l'algorithme de Newton-Euler du point de vue du temps nécessaire à effectuer un pas de calcul. Nous montrons ensuite le principe de séparation de modèle (et du code) séquentiel en plusieurs modèles pour la simulation en parallèle et en temps réel. Finalement, nous démontrons l'efficacité de l'utilisation d'un système multi-PC pour améliorer le temps de calcul de ces algorithmes.

5.1. Description du manipulateur et de son environnement

5.1.1. Structure du manipulateur

Le manipulateur modélisé est une main robotique. Cette main est composée de 3 doigts identiques à 3 articulations rotatives. Les doigts sont espacés d'un angle de 120° entre eux par rapport à l'axe vertical. La figure 5.1 présente un schéma de la main. La figure 5.2 présente la structure des référentiels d'un des doigts et les paramètres Denavit-Hartenberg modifiés d'un des doigts sont donnés au tableau 5.1.

Figure 5-1 : Schéma de la main robotique modélisée

Nous calculons la cinématique inverse par la méthode géométrique. Cette structure de manipulateur possède deux configurations possibles. La première, que nous n'utiliserons pas, et pour laquelle l'angle θ_2 est négatif produirait possiblement des collisions non modélisées entre les doigts (superposition des membres de différents doigts).

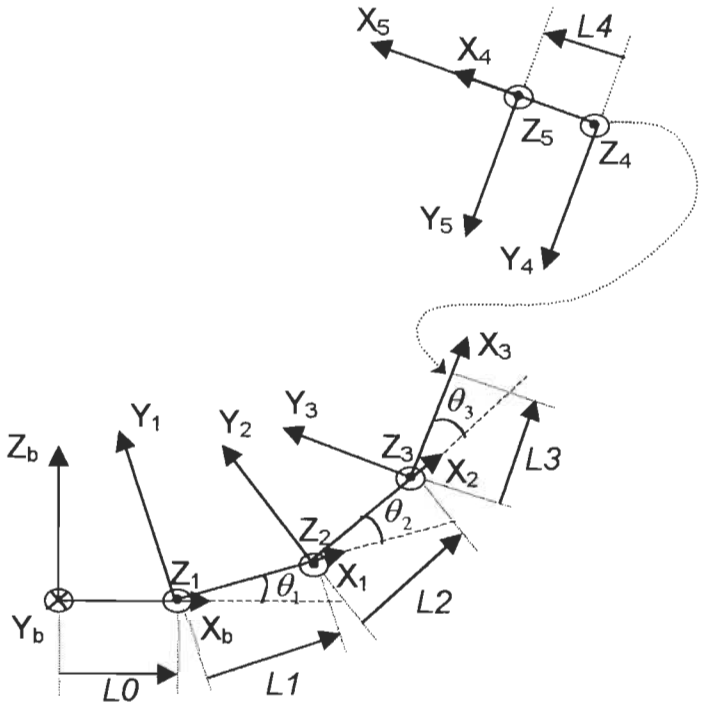

Figure 5-2 : Structure géométrique d'un doigt de la main modélisée

Tableau 5-1 : Paramètres D-H modifiés d'un doigt

# d'articulation	α_{i-1}	a_{i-1}	d_i	θ_i
1	$\pi/2$	$L0 = 0.2$m	0	θ_1
2	0	$L1 = 0.8$m	0	θ_2
3	0	$L2 = 0.8$m	0	θ_3
4	0	$L3 = 0.6$m	0	$\pi/2$
5	0	$L4 = 0.1$m	0	0

Nous choisissons donc le cas pour lequel θ_2 est positif. La figure 5.3 montre un schéma d'un doigt du premier au troisième référentiel. Les détails des calculs de la cinématique inverse sont présentés en annexe E.

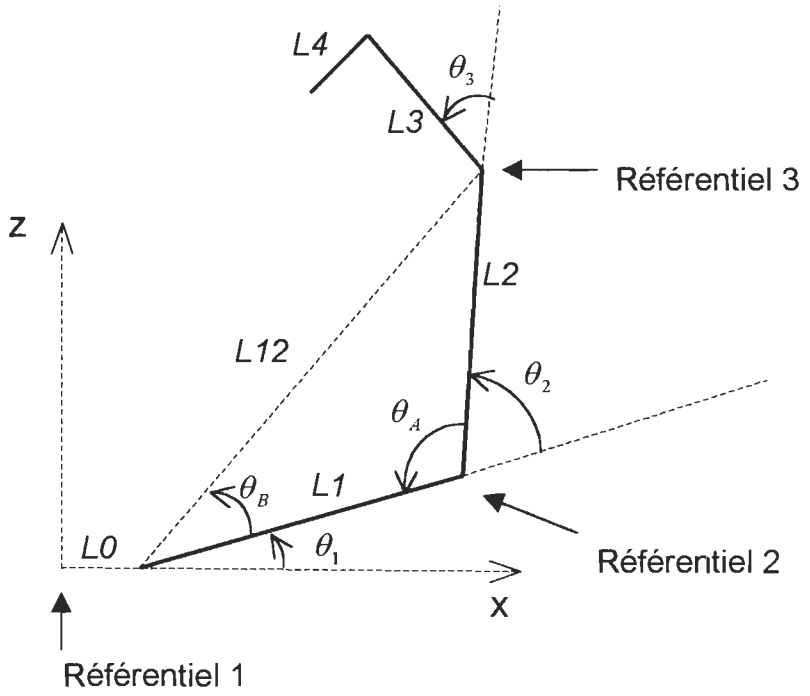

Figure 5-3 : Schéma de calcul de la cinématique inverse

Les résultats sont les suivants :

$$\theta_2 = \pi - \arccos\left(\frac{L2^2 + L1^2 - (p3x - L0)^2 - p3z^2}{2\,L1\,L2}\right) \tag{5.1}$$

$$\theta_1 = \arctan\left(\frac{p3z}{p3x - L0}\right) - \frac{\theta_2}{2} \tag{5.2}$$

$$\theta_3 = \theta_{123} - \theta_1 - \theta_2 \tag{5.3}$$

où $p3x$ est la position en x du référentiel 3 dans le référentiel 1, $p3z$ est la position en z du référentiel 3 dans le référentiel 1, θ_{123} caractérise l'orientation de l'effecteur (l'orientation de l'effecteur $\theta_e = \theta_{123} + \pi/2$ sur la figure 5.3).

5.1.2. Modèle de l'environnement

L'objet manipulé par la main robotique est un pentaèdre triangulaire comme montré à la figure 5.4.

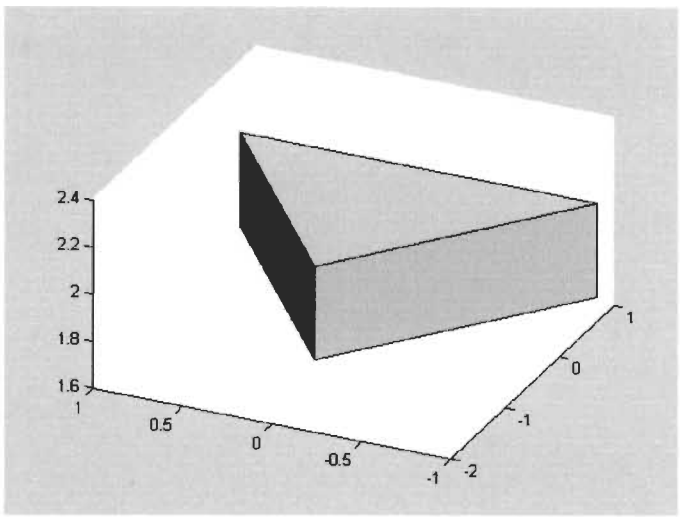

Figure 5-4 : Objet manipulé par la main robotique

Les parois de cet objet sont supposées flexibles, c'est à dire que si nous appliquons une force sur une paroi, alors elle se déformera proportionnellement et dans la direction de la force. Notons que seules les parois de l'objet sont permissives ; l'effecteur des doigts est rigide. La position de l'objet en z (par rapport au référentiel de la base du doigt 1, qui est considéré comme le référentiel de la base de la main entière) est toujours égale à 1.9m. De plus, l'objet est fixe en rotation (par exemple, un peu comme le bras de transmission manuelle d'une automobile). La hauteur de l'objet est de 0.4m. Le centre de l'objet est initialement situé en (x,y,z)=(0,0,1.9). La distance entre le centre de l'objet et chacune des 3 surfaces est de 0.7m. L'objet est caractérisé par une densité uniforme et sa masse est de 5kg. De plus, cet objet se déplace avec un certain frottement. Nous posons que le

frottement est principalement caractérisé par un frottement visqueux d'une valeur de 10 Ns/m.

Le doigt 1 approche l'objet en positionnant l'axe x de son effecteur X_5 (fig 5.2) parallèle à l'axe x de la base de la main X_b (et du doigt 1). Les doigts 2 et 3 approchent l'objet respectivement dans le sens négatif et positif de l'axe y de la base de la main (la trajectoire comprend aussi une composante en x, positive dans les deux cas).

Lorsque les doigts touchent à l'objet, la première étape est de calculer la résultante de force. Nous supposons que les doigts sont munis d'un capteur de force tactile placé au bout de l'effecteur (et orienté vers l'axe x de l'effecteur). Nous posons que la déformation du doigt et du capteur de force sont négligeables. De plus, nous supposons que les surfaces de l'objet sont toujours perpendiculaires aux effecteurs des doigts. En ce qui concerne le doigt 1, la tâche est simple ; la force en x est directement la force mesurée au bout du doigt 1 (car le référentiel du doigt 1 est identique à celui de l'objet). Pour le doigt 2, la force captée par rapport au référentiel de l'environnement (ou du doigt 1) est calculée par

$$\begin{aligned} F_{2x} &= F\sin\left(\pi/6\right) \\ F_{2y} &= F\cos\left(\pi/6\right) \end{aligned} \tag{5.4}$$

où F_{2x} est la contribution en force du doigt 2 en x par rapport à la base de l'environnement ; F_{2y} est la contribution en force du doigt 2 en y par rapport à la base de l'environnement ; F est la force (normale à la surface) captée par le doigt 2

Pour le doigt 3, la force captée par rapport au référentiel de l'environnement est obtenue par

$$\begin{aligned} F_{3x} &= F\cos\left(\pi/6\right) \\ F_{3y} &= F\sin\left(\pi/6\right) \end{aligned} \tag{5.5}$$

Nous effectuons ensuite la sommation des forces en x et en y. Après avoir calculé la résultante de force appliquée sur l'objet, nous trouvons la position centrale de l'objet (par rapport au référentiel de l'objet). Le comportement de l'objet (en x et en y) peut être calculé à partir de :

$$F_{row} = M\frac{d^2w}{dt^2} + F_v\frac{dw}{dt} \tag{5.6}$$

où w est la position de l'objet (w mesuré dans la direction x ou y) ; F_{row} est la force appliquée par le robot sur l'objet, dans la direction w ; M est la masse de l'objet ; F_v est le frottement visqueux.

Après quelques manipulations et entrée des valeurs numériques, nous obtenons le modèle d'état

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -2 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.2 \end{bmatrix}u \tag{5.7}$$

où x_1 est la position de l'objet (en x ou en y) ; u est la force totale appliquée sur l'objet (aussi en x ou en y).

Notons que nous devons appliquer (5.7) une fois en x et une fois en y. Lorsque l'objet bouge, alors les frontières définissant l'objet vu par les trois doigts bougent elles aussi. Les équations permettant de calculer la position de ces frontières sont

$$
\begin{aligned}
px_{1o} &= px + 0.7 \\
py_{1o} &= py \\
px_{2o} &= -\cos\left(\pi/3\right) px + \sin\left(\pi/3\right) py + 0.7 \\
py_{2o} &= -\sin\left(\pi/3\right) px - \cos\left(\pi/3\right) py \\
px_{3o} &= -\cos\left(\pi/3\right) px - \sin\left(\pi/3\right) py + 0.7 \\
py_{3o} &= \sin\left(\pi/3\right) px - \cos\left(\pi/3\right) py
\end{aligned}
\tag{5.8}
$$

où px_{1o} est la position limite en x (axe du référentiel du doigt 1) de l'objet ; py_{1o} est la position limite en y (axe du référentiel du doigt 1) de l'objet ; px_{2o} est la position limite en x (axe du référentiel du doigt 2) de l'objet ; py_{2o} est la position limite en y (axe du référentiel du doigt 2) de l'objet ; px_{3o} est la position limite en x (axe du référentiel du doigt 3) de l'objet ; py_{3o} est la position limite en y (axe du référentiel du doigt 3) de l'objet ; px est la position en x du centre de l'objet par rapport au référentiel de l'environnement ; py est la position en y du centre de l'objet par rapport au référentiel de l'environnement.

En résumé, nous calculons la force reçue par l'objet provenant de chacun des doigts. Ensuite, nous effectuons une sommation des trois forces en x et en y. Par la suite, nous calculons l'équation de mouvement du centre de l'objet pour en calculer la nouvelle position. Finalement, nous calculons la position des trois surfaces de contact (une surface par doigt). Notons que chacune de ces surfaces est définie en x et en y (chaque doigt possède son propre référentiel, identique à celui du doigt 1 montré à la figure 5.2) ; la coordonnée x varie la proximité de la surface avec le doigt tandis que la coordonnée y est contrôlable seulement par les doigts opposés au doigt du référentiel dont on doit calculer la coordonnée y. Nous pouvons comprendre que si un doigt pousse l'objet trop loin, alors l'objet ne pourra pas être saisi par les autres doigts.

Les tests effectués pour savoir s'il y a contact (calcul effectué dans le bloc de ressort équivalent de l'environnement et dans le bloc du modèle de l'objet) sont résumés ci-dessous

Conditions : $P_{ex} \leq P_{xo}$ (5.9)

$$(P_{yo} - 1.2124) \leq P_{ey} \leq (P_{yo} + 1.2124) \quad (5.10)$$

où P_{ex} est la position de l'effecteur du doigt en x ; P_{xo} est la position en x de l'objet ; P_{yo} est la position en y de l'objet ; P_{ey} est la position de l'effecteur en y (nulle).

5.1.3. Programmes de visualisation des résultats

Nous avons conçu un programme de visualisation des résultats en trois dimensions permettant de voir la géométrie de la main en fonction du temps. Ceci permettra une analyse plus complète et avec plus d'intuition des variables articulaires qu'en regardant des graphiques des variables articulaires en fonction du temps. L'animation en 3 dimensions a été conçue sous forme d'une « s-function » de *Matlab*® ; cette fonction est présentée en annexe F et peut fonctionner en temps réel ou différé. Les principales fonctions du programme d'animation sont présentées à la figure 5.5.

À l'initialisation, le programme ajuste l'élévation, l'azimuth et d'autres paramètres nécessaires à la construction de la figure du tracé. Le programme calcule ensuite les matrices des vertices (points) définissant chacun des membres de la main (sous une forme utilisable pour l'affichage). Les origines des référentiels des membres sont initialement toutes placées à l'origine du référentiel de la base de la main (la base de la main est superposée à la base du premier doigt). Ensuite, le programme transforme les matrices de vertices sous une forme optimisée pour le calcul utilisant des matrices de transformation. Le programme calcule ensuite la position actuelle des vertices en utilisant des matrices de transformation (provenant des équations de la cinématique directe). Le programme réorganise les matrices de vertices pour l'affichage et il effectue un tracé invisible de la main.

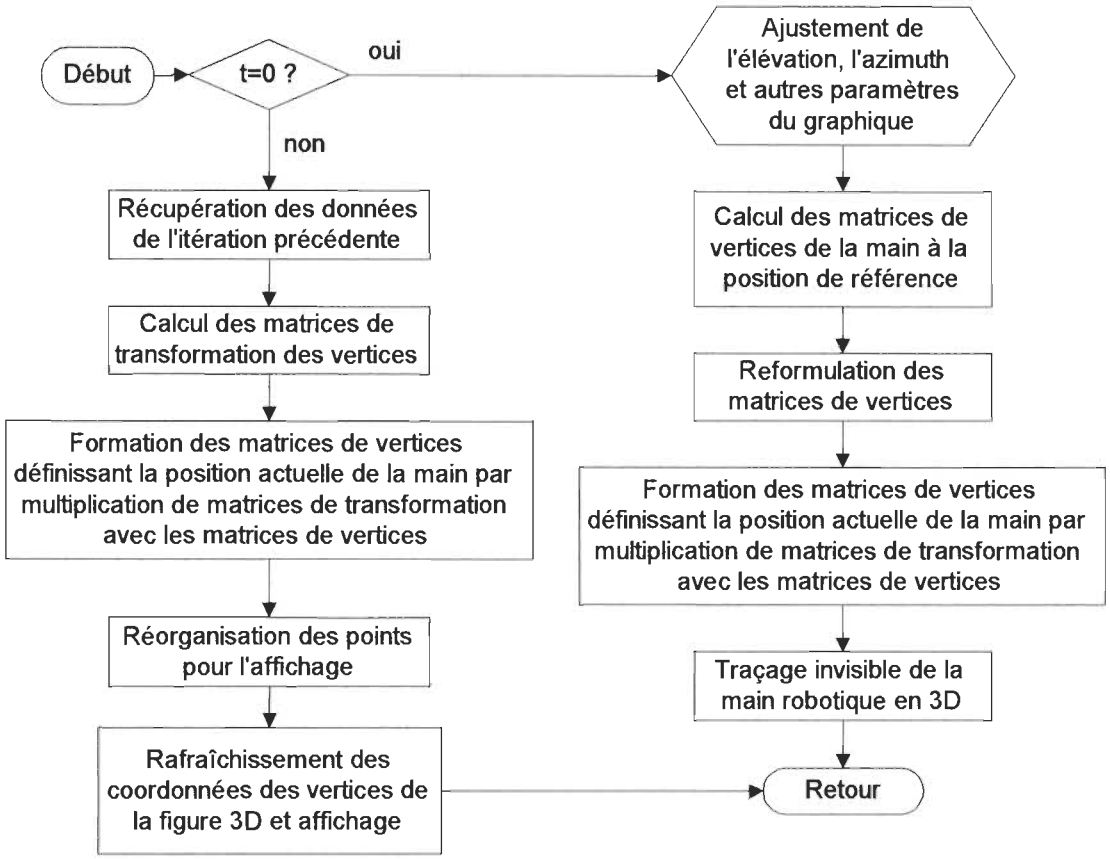

Figure 5-5 : Fonctionnement du programme d'animation en trois dimensions "sfun3d.m"

La procédure d'affichage pour chaque itération subséquente est la suivante. Premièrement, le programme récupère les données de l'itération précédente (coordonnées des vertices des membres à l'origine). Ensuite, il calcule les matrices de transformation des vertices à partir des variables articulaires actuelles. Le programme utilise ces matrices de transformation pour calculer la position actuelle des vertices. Il réorganise les matrices de vertices pour l'affichage pour finalement rafraîchir les coordonnées du tracé (en le rendant visible).

5.2. Main robotique commandée en position et en force

5.2.1. Schémas *Simulink*™ des modèles obtenus pour la formulation Lagrange-Euler

Nous avons utilisé le générateur de code automatique pour produire le modèle de la main robotique avec la formulation de Lagrange-Euler et le contrôleur de position utilisant l'algorithme de Slotine et Li. Les blocs d'appel des fonctions de calcul des éléments du modèle Lagrange-Euler sont toujours branchés de la même façon pour un robot série quelconque. La figure 5.6 montre le schéma *Simulink*™ général du modèle Lagrange-Euler de la main robotique. Sauf indication contraire, ces codes ne sont pas générés automatiquement en entier et requièrent principalement une intervention de l'utilisateur pour compléter certaines connexions.

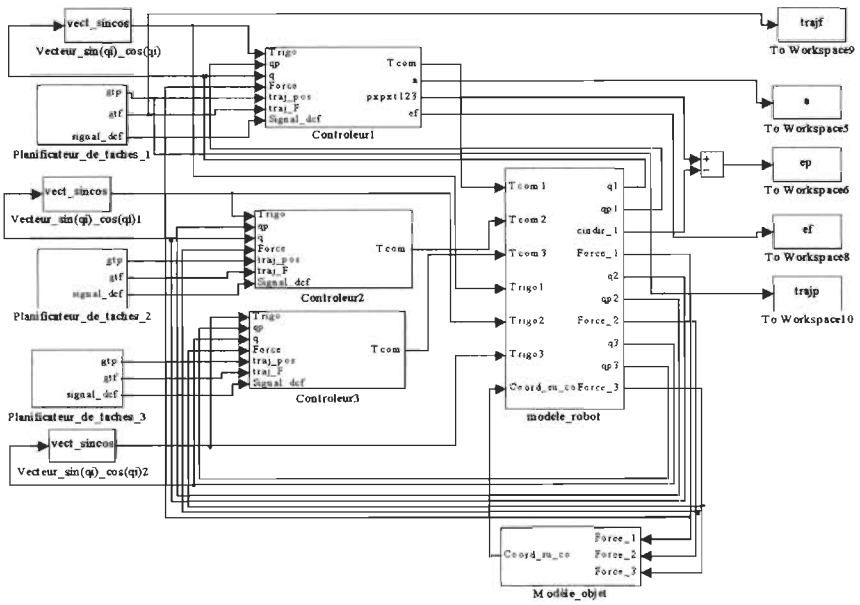

Figure 5-6 : Schéma général de simulation de la main sous la formulation de Lagrange-Euler

Les planificateurs de tâches contiennent les générateurs de trajectoires de position et de force pour les trois doigts ainsi que le bloc de supervision. Les blocs « contrôleur » contiennent les contrôleurs de position et de force, le calcul de la cinématique inverse, le calcul du Jacobien inverse et de sa dérivée. Le bloc « modele_robot » calcule la dynamique directe de la main (les trois doigts indépendants) et une partie de l'interaction entre les doigts et l'environnement. Le bloc « Vecteur_sin(qi)_cos(qi) » calcule le vecteur des termes trigonométriques nécessaires au calcul du modèle et du contrôleur de position. Le code appelé par ce bloc est une « s-function » en langage C. Le code de « Vecteur_sin(qi)_cos(qi) » est généré automatiquement. Finalement, le bloc « modele_objet » calcule la sommation des forces exercées sur l'objet de même que sa position.

La figure 5.7 montre l'intérieur d'un bloc de planificateur des tâches. Les entrées du planificateur de tâches sont l'horloge, la consigne de force, les positions initiales et finales des mouvements ainsi que leur durée. Les sorties sont le temps actuel, le temps final, les positions initiales et finales pour le contrôleur de position et les forces initiales et finales pour le contrôleur de force. Pour désactiver le contrôleur de force, nous posons le temps actuel (première entrée de « Gen_traj_F ») à zéro. La sortie « signal_dcf » sert à remettre volontairement l'intégrateur du contrôleur de force à zéro. La figure 5.8 montre l'intérieur du bloc des contrôleurs. Notons que la commande de force (contrôleur PI) s'effectue seulement selon la direction de l'axe x du référentiel du doigt. L'intégrateur du contrôleur PI de force est remis à zéro lorsque le doigt quitte l'objet ou lorsque le contrôleur de force est désactivé (signal_dcf).

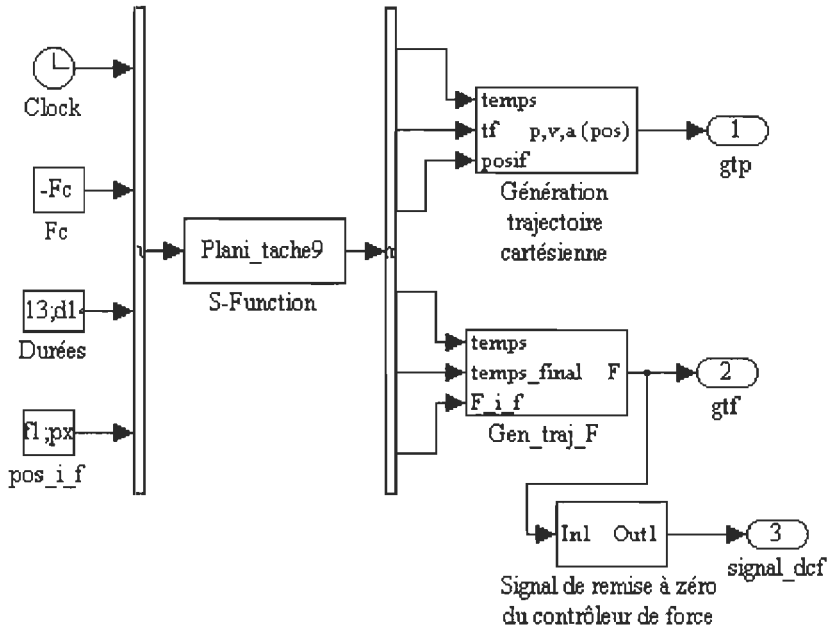

Figure 5-7 : Planificateur de tâches et générateurs de trajectoire

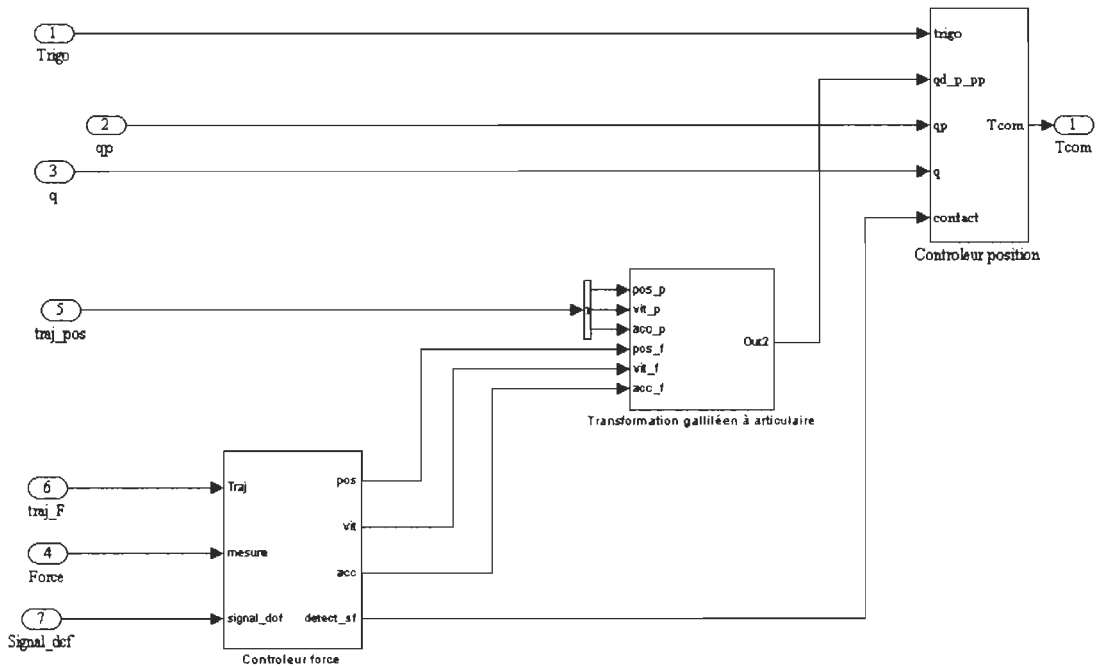

Figure 5-8 : Schéma Simulink™ du bloc des contrôleurs

Les générateurs de trajectoires et le contrôleur de force sont dans le domaine galliléen tandis que la commande de position s'effectue dans le domaine articulaire. Nous devons donc transformer la trajectoire de position totale à l'aide de la cinématique inverse, du Jacobien inverse et de sa dérivée.

La figure 5.9 montre le contrôleur de force. Les équations implantées dans le bloc « Contrôleur PI » (figure 5.10) sont présentés au chapitre III en (3.27) et (3.29). Les paramètres du contrôleur sont les suivants. Premièrement, le gain proportionnel est fixe et sa valeur est $kp = 0.3$. Les paramètres du gain intégral variable sont : k_0=0.01 (borne inférieure), $k_1 = 10$ (borne supérieure) et $k_2 = 10$ (sensibilité). Les fonctions à gauche du calcul de *ki* sur la figure 5.10 servent à déterminer le signe de l'erreur de force comme montré dans l'étude du contrôleur de force. Le bloc « switch1 » commute le signe de l'erreur de force lors du relâchement du contact comme montré en (3.29). Le détecteur de contact produit deux signaux (figure 5.9). Le premier sert à remettre l'intégrateur du contrôleur de force à zéro 10 périodes d'échantillonnage après que le doigt ait perdu le contact avec l'objet. Le second arrête l'identification des masses du contrôleur de position dès que le doigt entre en contact avec l'objet.

La sortie du contrôleur PI de force est une position. Nous avons besoin de la position, vitesse et accélération pour le contrôleur de position. Pour les obtenir, nous utilisons un filtre du deuxième ordre (deux pôles à 100 rad/sec) et des dérivées comme montré au chapitre III en (3.31).

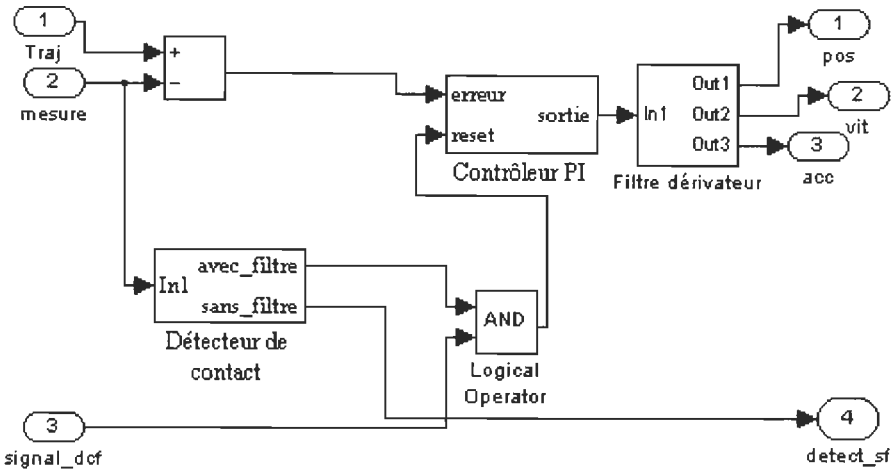

Figure 5-9 : Contrôleur de force et filtres

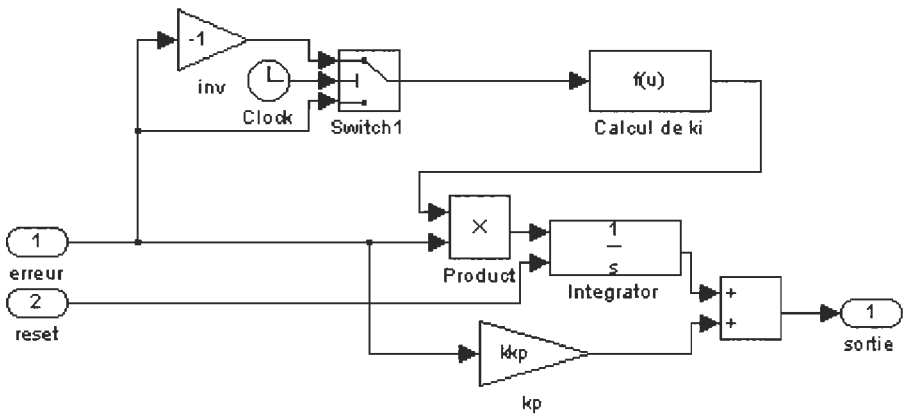

Figure 5-10 : Contrôleur PI non linéaire de force

Le schéma du filtre dérivateur est montré à la figure 5.11. Nous avons implanté le filtre dérivateur sous forme d'équation d'état pour faciliter l'ajustement des conditions initiales.

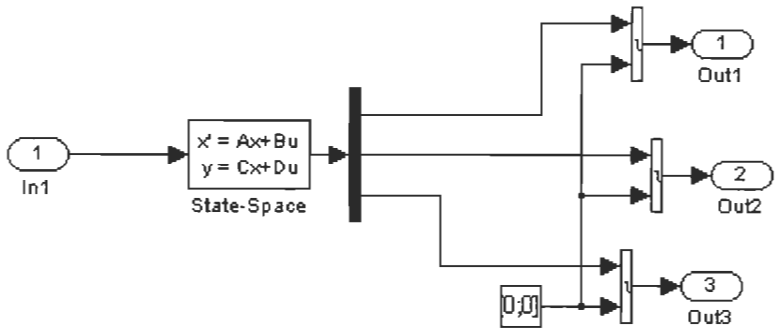

Figure 5-11 : Filtre dérivateur de sortie du contrôleur de force

Le schéma du détecteur de contact est présenté à la figure 5.12. Nous supposons que le capteur de force est idéal (qu'il détecte des forces infiniment petites). La sortie du détecteur de contact est passée dans un filtre logique permettant de déterminer s'il y a eu une perte de contact pendant au moins 10 pas d'échantillonnage consécutifs avant de remettre l'intégrateur à zéro. Ce filtre sert à empêcher la destruction d'efforts de commande utiles lors de rebondissements pouvant se produire au début d'une tâche de contact.

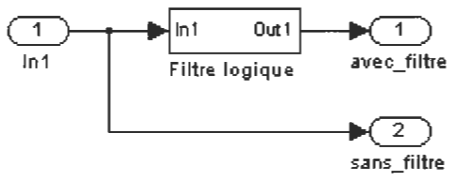

Figure 5-12 : Schéma Simulink™ du détecteur de contact

Le filtre logique est présenté à la figure 5.13. Notons que nous utilisons un détecteur de contact sans filtre logique pour la désactivation de l'identification des paramètres.

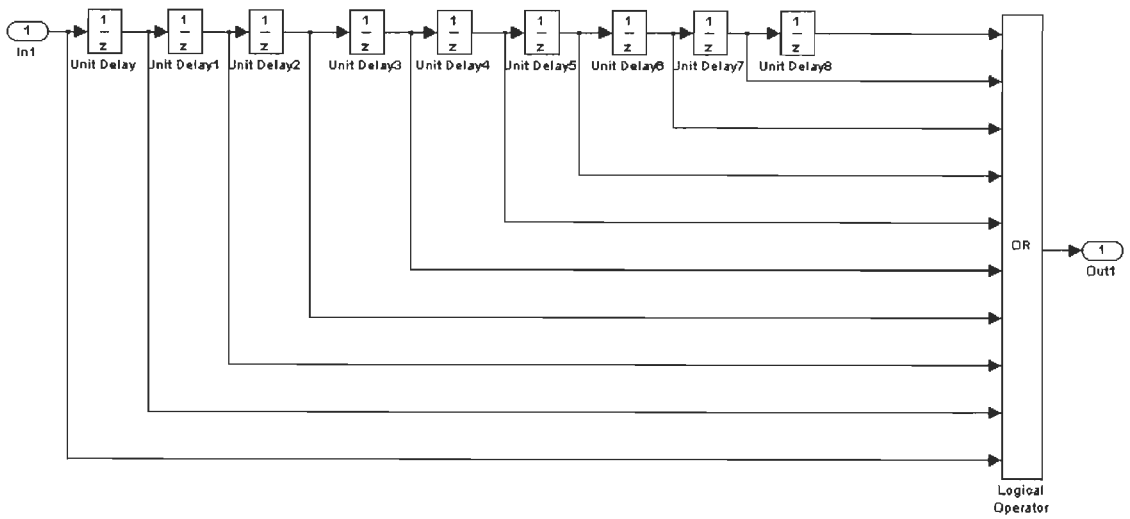

Figure 5-13 : Schéma Simulink™ du filtre logique

La figure 5.14 montre le bloc de transformation de la trajectoire de position du domaine galliléen au domaine articulaire. Les blocs « Cin_inv_RRR » et « Jacobien vitesse accélération » calculent la cinématique inverse et l'inverse généralisé du Jacobien de vitesse et sa dérivée. Les équations de la cinématique inverse sont présentées au chapitre V en (5.1), (5.2) et (5.3). Les équations du calcul des vitesses et accélérations articulaires en fonction des vitesses et accélérations galliléennes sont présentées au chapitre III de (2.4) à (2.7). Le code du calcul des vitesses et accélérations articulaires est généré automatiquement sous forme de « s-function » en langage C. Les entrées de la cinématique inverse sont les positions de consigne en x (corrigée par le contrôleur de force) et en z de l'effecteur de même que son orientation par rapport à l'axe x du doigt (sommation des trois angles à trouver). La sortie représente la trajectoire de position articulaire de référence. Les entrées du bloc de calcul de vitesses et accélérations articulaires sont les vitesses et accélérations cartésiennes et la cinématique inverse (positions angulaires des membres).

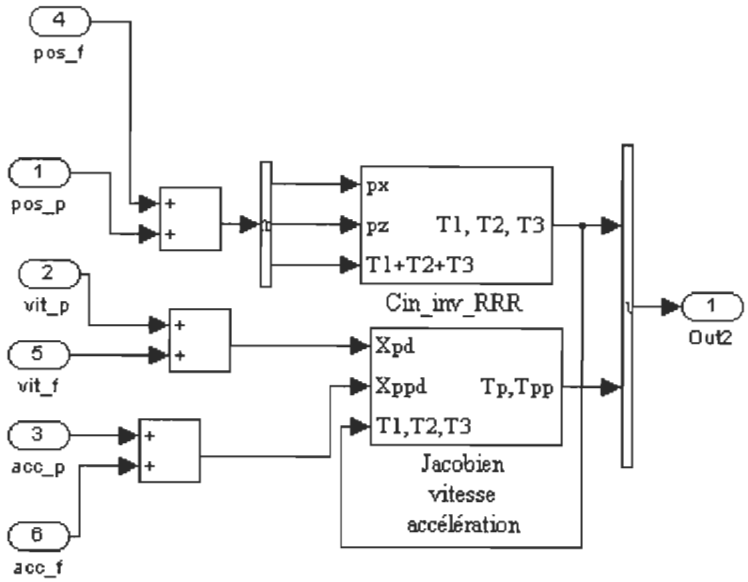

Figure 5-14 : Transformation de la trajectoire de position du domaine galliléen au domaine articulaire

Nous pouvons voir le schéma *Simulink*™ du contrôleur adaptatif de position à la figure 5.15 (voir aussi figure 3.1). Les intégrateurs compris dans le code de la loi de commande adaptative sont continus. Vu que le pas de calcul est fixe et que l'algorithme d'intégration utilisé par Simulink (Runge-Kutta ODE4) est classique, il est possible d'implanter cette loi de commande de la même façon que si les intégrateurs étaient discrets.

Le bloc « Calcul_qr_s » calcule $\dot{\mathbf{q}}_r$ et $\ddot{\mathbf{q}}_r$, l'erreur de suivi « $\mathbf{s}$ » et le signal $\ddot{\mathbf{q}}_r - \mathbf{s}\boldsymbol{\Lambda}$ permettant d'inclure implicitement le contrôleur PD. La figure 5.16 présente ce bloc. Les paramètres du contrôleur sont les suivants : $\mathbf{P}_0 = \mathbf{I}$ (conditions initiales de $\mathbf{P}$), $\boldsymbol{\Lambda}$=[100;100;100] (rigidité du contrôleur de position et rapidité de l'élimination de

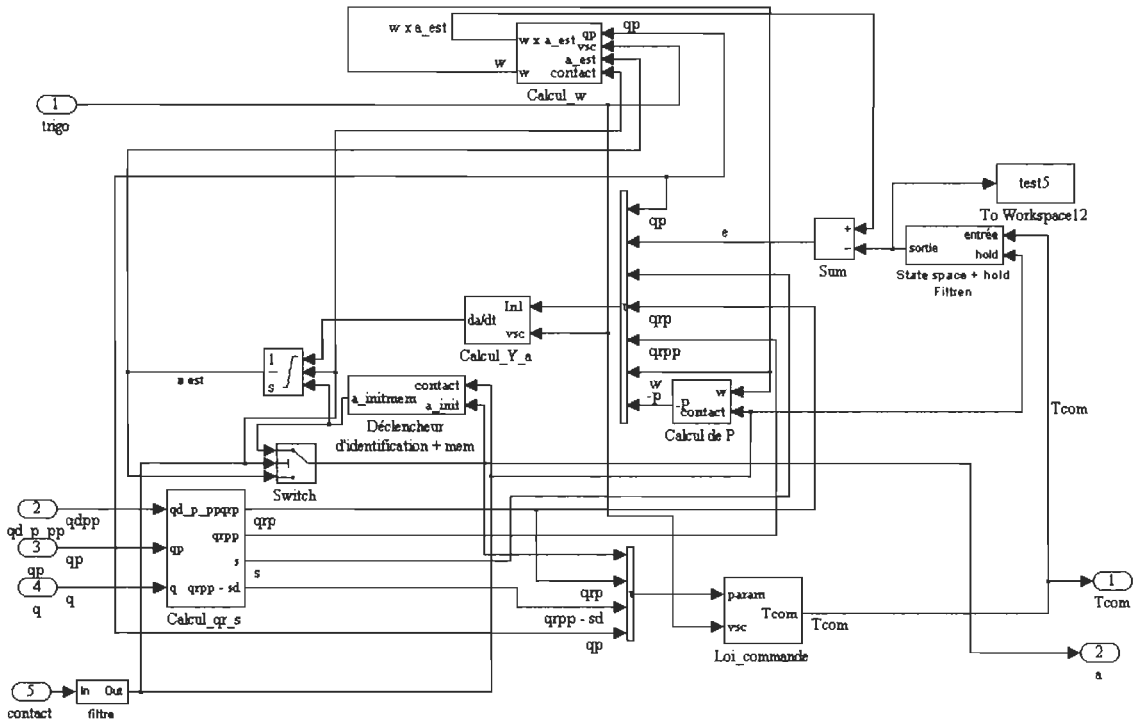

Figure 5-15 : Schéma Simulink™ du contrôleur adaptatif de position

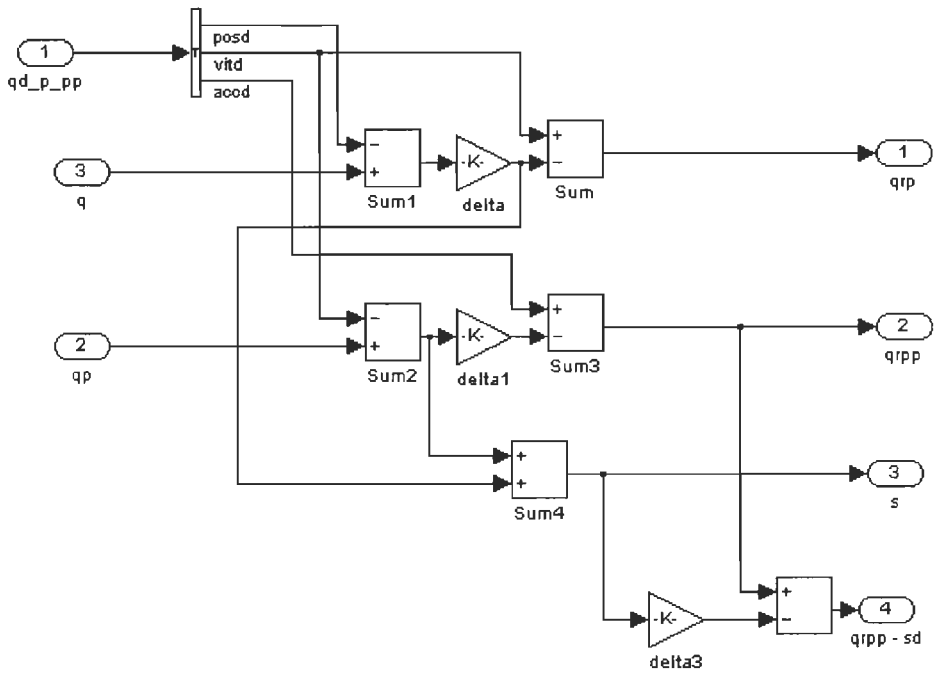

Figure 5-16 : Bloc de calculs divers du contrôleur de position

l'erreur en régime permanent), $\lambda = 0.2$ (filtre de **W** et du couple de commande), $\rho = 0.5$ (facteur d'oubli), $k_{\lim} = 1000$ (limite supérieure du module de **P**), $\mathbf{R} = 500\,\mathbf{I}$.

Le code de la loi de commande est généré automatiquement sous forme de « s-function » en langage C à partir de l'équation (3.6). Les entrées de la loi de commande sont les paramètres estimés (les masses), les vitesses et accélérations articulaires, la constante de gravité et le vecteur de données trigonométriques. Le Schéma de la loi de commande est présenté à la figure 5.17. Les paramètres Denavit-Hartenberg modifiés constants sont automatiquement inclus dans les codes du contrôleur et du modèle du robot sous forme de constantes.

Figure 5-17 : Bloc de la loi de commande en position

Le bloc « Déclencheur d'identification + mem » sert à conserver constants les paramètres estimés lors d'un contact entre le robot et un objet (lorsque l'identificateur est désactivé). L'intérieur du bloc est présenté à la figure 5.18.

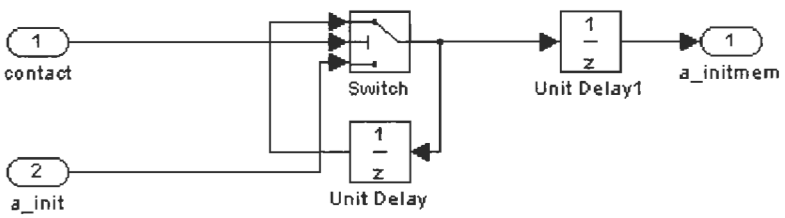

Figure 5-18 : Bloc de mémorisation des paramètres estimés

Le bloc de calcul de « **-P** » est présenté à la figure 5.19. Le code du calcul de $\dot{\mathbf{P}}$ est généré automatiquement sous forme de « s-function » en langage C à partir de l'équation 3.14. Les entrées de la fonction de calcul de « **-P** » sont la matrice **W**, la valeur précédente de **P**, le facteur d'oubli ρ et la borne supérieure du module de **P**. Nous conservons aussi la valeur de **P** constante durant un contact pour éviter une divergence.

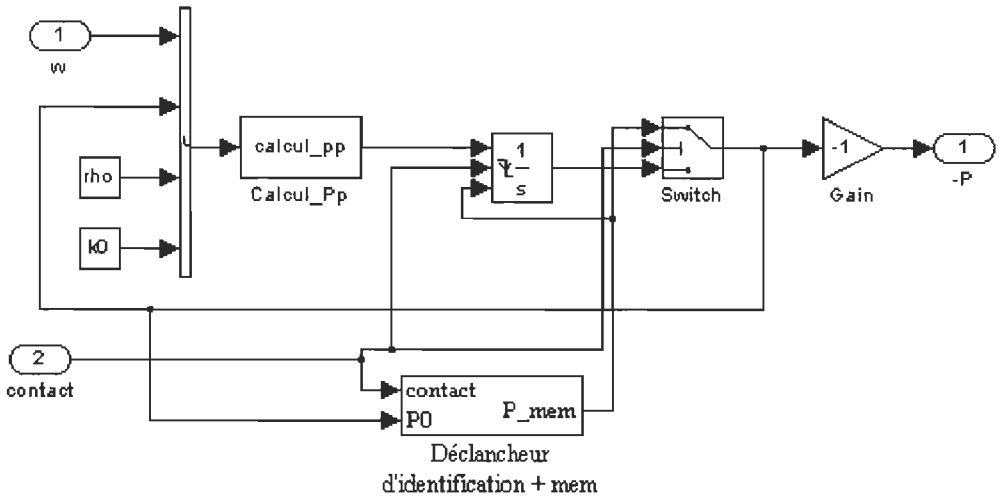

Figure 5-19 : Bloc de calcul de -P

Nous présentons à la figure 5.20 le schéma *Simulink*™ du bloc de calcul de la loi d'adaptation (calcul de la dérivée des paramètres estimés). Ce bloc calcule $\dot{\mathbf{a}}$ à partir de l'équation 3.13. Ce code est généré automatiquement sous forme de « s-function » en langage C. Les entrées sont les vitesses $\dot{\mathbf{q}}$, $\dot{\mathbf{q}}_r$ et accélération $\ddot{\mathbf{q}}_r$ articulaires, l'erreur de prédiction **e**, l'erreur de suivi **s**, la matrice des signaux **W** , l'opposé de la matrice de covariance **-P**, la matrice de poids d'importance de l'erreur de prédiction **R**, la constante de gravité *g* et le vecteur des termes trigonométriques.

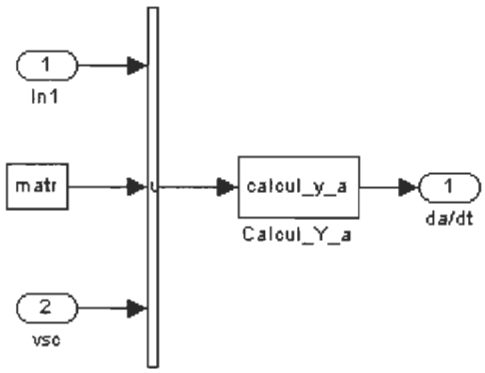

Figure 5-20 : Schéma Simulink™ du bloc de calcul de la loi d'adaptation des paramètres

La figure 5.21 montre le schéma du bloc de calcul de la matrice des signaux **W**. Nous calculons **W** à partir de (3.10). Le code du calcul de **W** est généré automatiquement sous forme de « s-function » en langage C. Les entrées sont la vitesse articulaire $\dot{\mathbf{q}}$, la gravité **g**, le vecteur des termes trigonométriques, le pôle du filtre λ et les masses estimées $\hat{\mathbf{a}}$. Les blocs « Prod_mat_vect » proviennent de la librairie de calcul matriciel et de robotique. L'entrée « hold » des filtres sert à maintenir les états constants lors de la désactivation de l'identificateur des paramètres.

Le bloc « modele_robot » (dans le schéma principal de simulation de la figure 5.6) calcule la dynamique directe des doigts (position, vitesse et accélération articulaires en fonction du couple articulaire) et une partie de l'interaction entre la main et l'environnement. Ce bloc est présenté à la figure 5.22. Les entrées sont les couples de commande articulaires « $\boldsymbol{\tau}$ » et « Coord_su_co », la coordonnée des surfaces de l'objet utile à la détection de contact et au calcul de la force exercée par l'effecteur. Les forces et

couples externes appliqués à l'effecteur « **FeNe** » sont calculés dans le modèle du ressort équivalent, la sortie « Force » étant la force de contact.

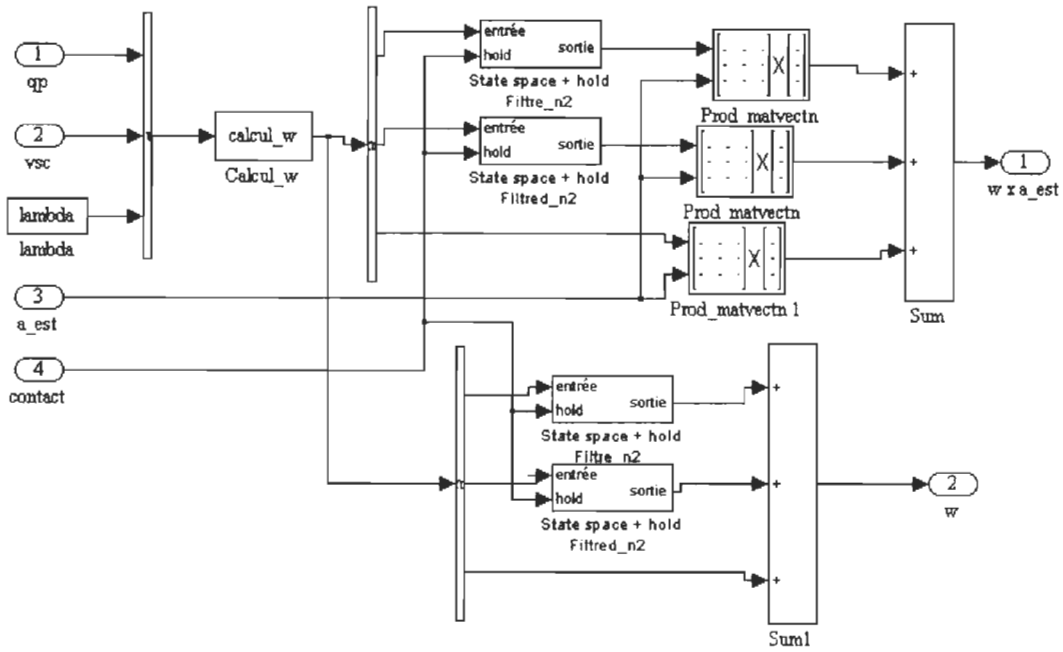

Figure 5-21 : Schéma Simulink™ du calcul de la matrice W

La figure 5.23 présente le schéma de calcul des forces et couples externes exercés par l'effecteur sur l'objet. Le calcul de la cinématique directe est effectué à partir de (2.1). Le Jacobien de force est la transposée du Jacobien de vitesse qui est calculé par (2.2) et (2.3). À partir de la cinématique directe et des coordonnées des surfaces de l'objet, ce bloc détecte s'il y a contact entre l'effecteur du robot et l'objet. S'il y a contact, le bloc calcule la force exercée par l'effecteur sur l'objet. Ensuite, par une fonction C de calcul du Jacobien de force, la force articulaire résultante est calculée. La fonction « détecteur de contact » vérifie si les équations (5.9) et (5.10) sont vraies.

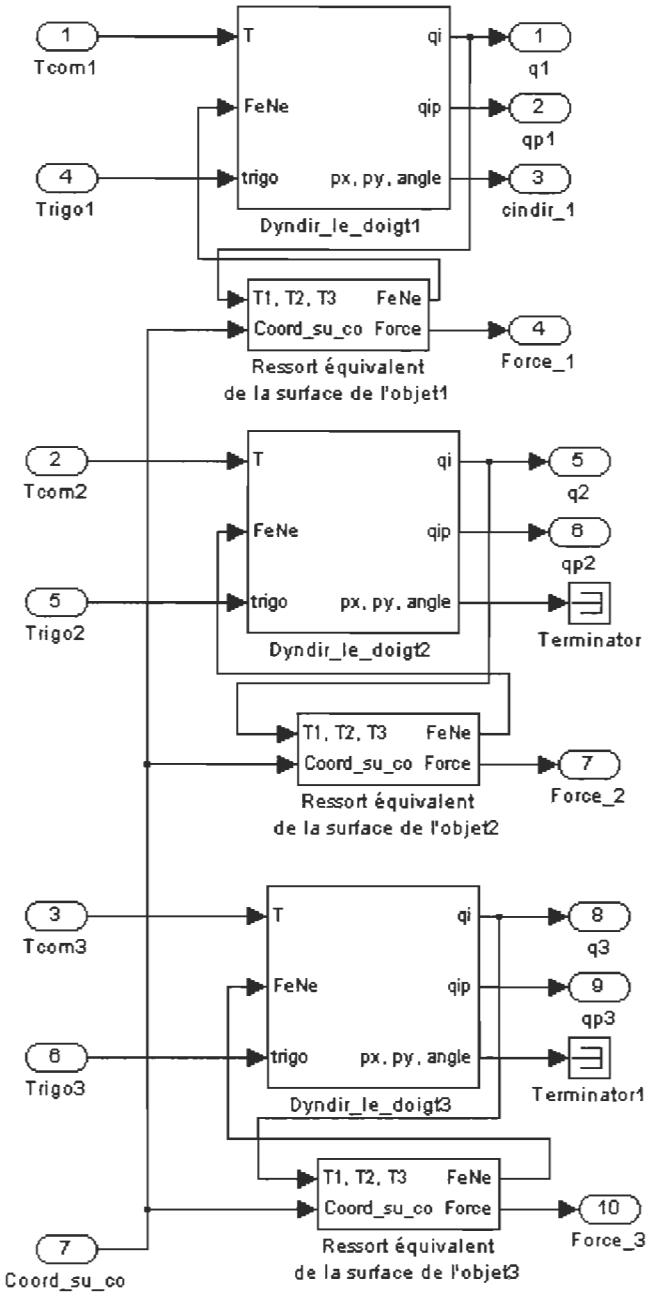

Figure 5-22 : Schéma Simulink™ du modèle de la main et de son interaction avec l'environnement

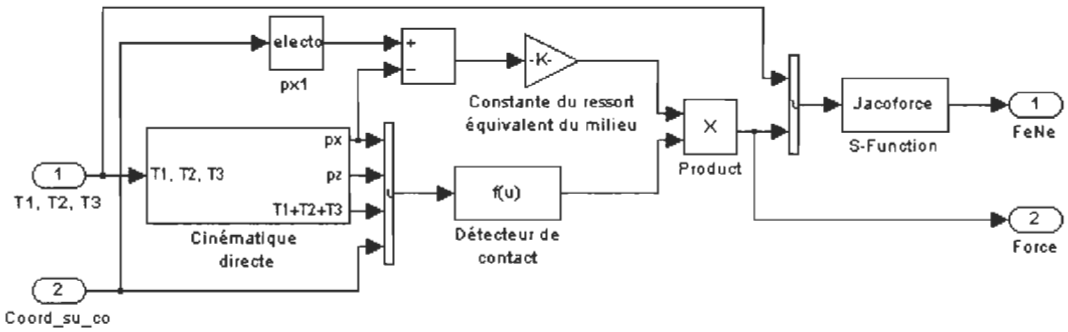

Figure 5-23 : Schéma Simulink™ de calcul de la force exercée par l'effecteur

Le bloc de calcul de la cinématique directe est présenté à la figure 5.24. L'entrée est le vecteur de variables articulaires et les sorties sont les positions en x et en z et l'orientation de l'effecteur θ_e (angle de l'axe x de l'effecteur par rapport à l'axe z de la base du doigt).

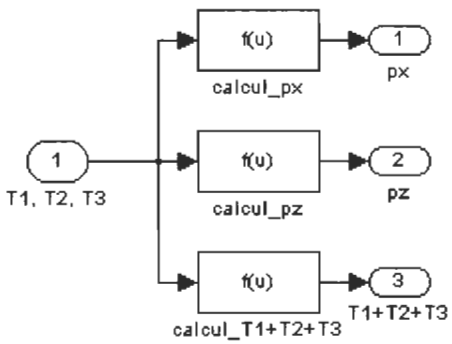

Figure 5-24 : Schéma *Simulink*™ du calcul de la cinématique directe

Chaque bloc de calcul de la dynamique directe contient le schéma *Simulink*™ présenté à la figure 5.25. La fonction « dyndirle_vg.dll » calcule le vecteur de termes centrifuges, de Coriolis et de gravité. La fonction « dyndirle_m.dll » calcule les éléments de la

matrice des masses. Le calcul des équations de ces codes a été effectué par les équations (2.8) à (2.23) mais les équations sont implantées selon la formulation de Lagrange-Euler (2.29). Ces codes sont générés automatiquement sous la forme de « s-function » en langage C. Dans ce modèle, nous considérons que seulement la masse de l'effecteur (masse 3) peut varier.

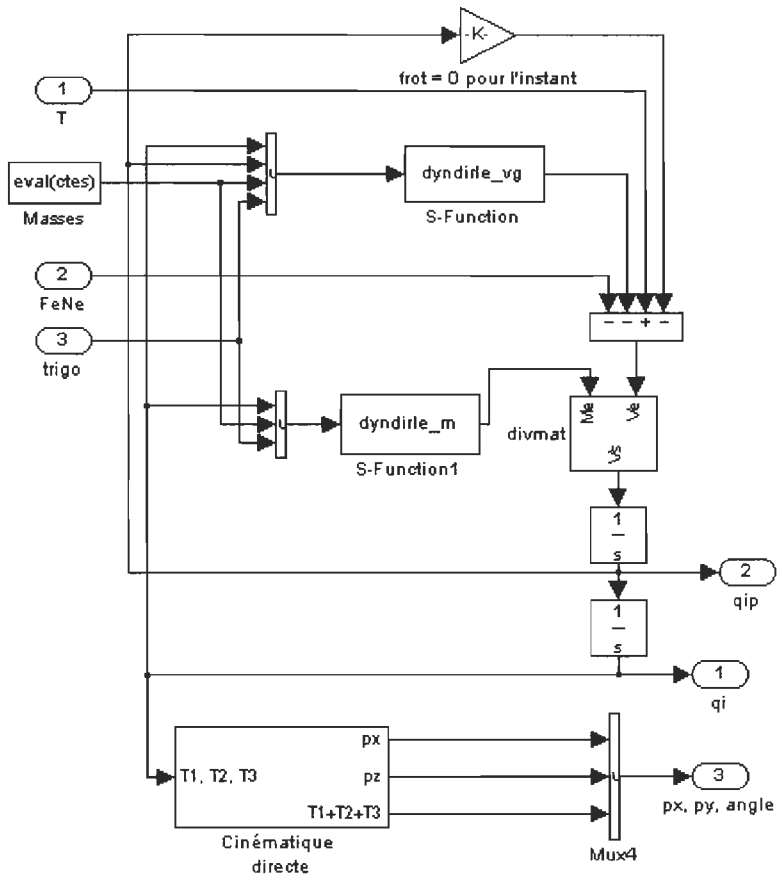

Figure 5-25 : Schéma Simulink™ du modèle de la dynamique d'un doigt

La figure 5.26 montre le modèle de l'environnement (l'objet en forme de pentaèdre) situé dans le bloc « modele_objet » montré à la figure 5.6. Les entrées sont les forces de contact des trois doigts sur l'objet et la sortie est la coordonnée des surfaces de l'objet (obtenue par la sous-routine C « Calcul_coord_lim.dll »). La somme des forces est

calculée à partir de (5.4) et (5.5). L'équation (5.7) permet de calculer (par une équation d'état) la position de l'objet en x et en y à partir de la sommation des forces. La fonction « Calcul_coord_lim » est produite à partir de (5.8).

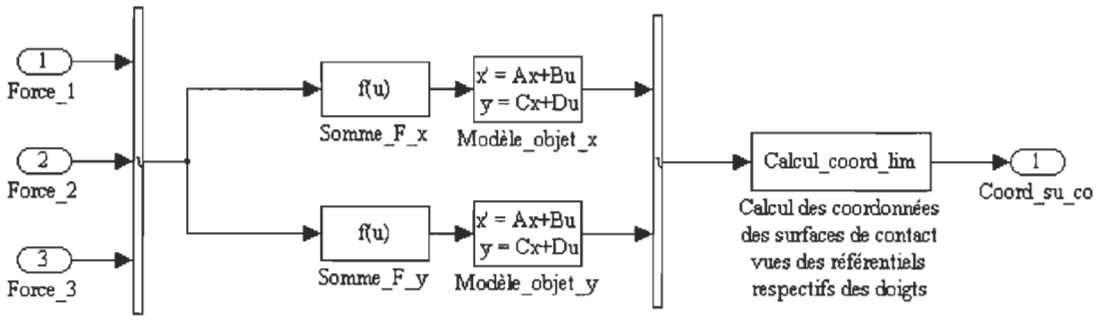

Figure 5-26 : Schéma Simulink™ du bloc du modèle de l'objet

5.2.2. Schémas *Simulink*™ des modèles obtenus par la formulation de Newton-Euler

Nous avons modélisé la cinématique et la dynamique directes d'un des doigts de la main robotique à l'aide du générateur de code automatique et en utilisant l'algorithme et la formulation récursive de Newton-Euler. Le schéma général du modèle d'un doigt est présenté à la figure 5.27. Les blocs « Cin_dir » et « dyndir_doigt » servent à calculer la cinématique et la dynamique directes du doigt. Le bloc « Entrées » contient les position, vitesse et accélération de la base du doigt, le poids, la position et la forme des masses de même que les paramètres Denavit-Hartenberg modifiés du doigt.

La figure 5.28 présente l'intérieur du bloc de cinématique directe. Chacun des blocs « cin_dir_art » calcule la matrice de rotation et le vecteur de translation reliant son

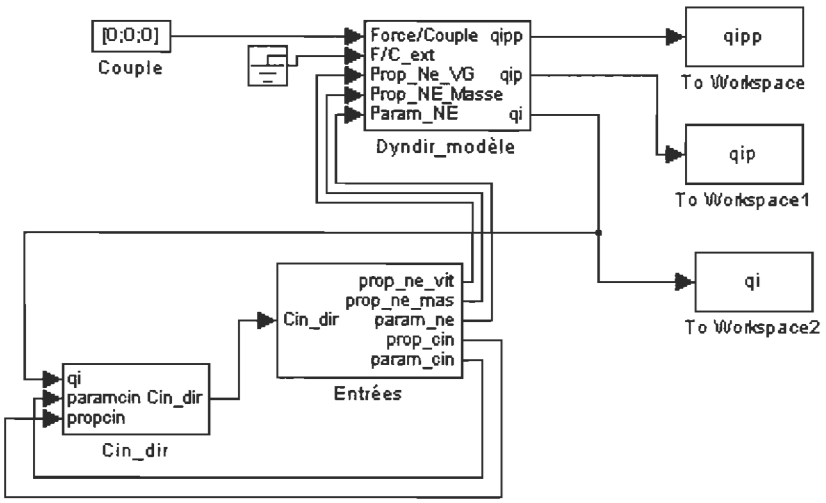

Figure 5-27 : Schéma général du modèle d'un doigt produit à l'aide de l'algorithme et de la formulation de Newton-Euler

référentiel au référentiel de l'articulation précédente. Les entrées sont les positions articulaires « qi », les paramètres D-H modifiés du doigt « paramcin » et la position de la base « propcin ». Les blocs de cinématique directe articulaire sont générés automatiquement sous forme de « s-function » en langage C. Les équations implantées dans ces codes sont présentés en (2.1).

La figure 5.29 montre l'intérieur du bloc « Entrées ». Le vecteur « Prop_ne_vit » contient les vitesses et accélérations de la base. Ce vecteur tient compte de la gravité (en utilisant le principe expliqué en (2.23)) car il est utilisé dans le calcul des termes de Coriolis, forces centrifiges et gravité. Le vecteur « Prop_ne_mas » contient aussi les vitesses et accélérations de la base mais sans tenir compte de la gravité car ce vecteur sert au calcul des éléments de la matrice des masses/inerties. Le vecteur « Param_ne »

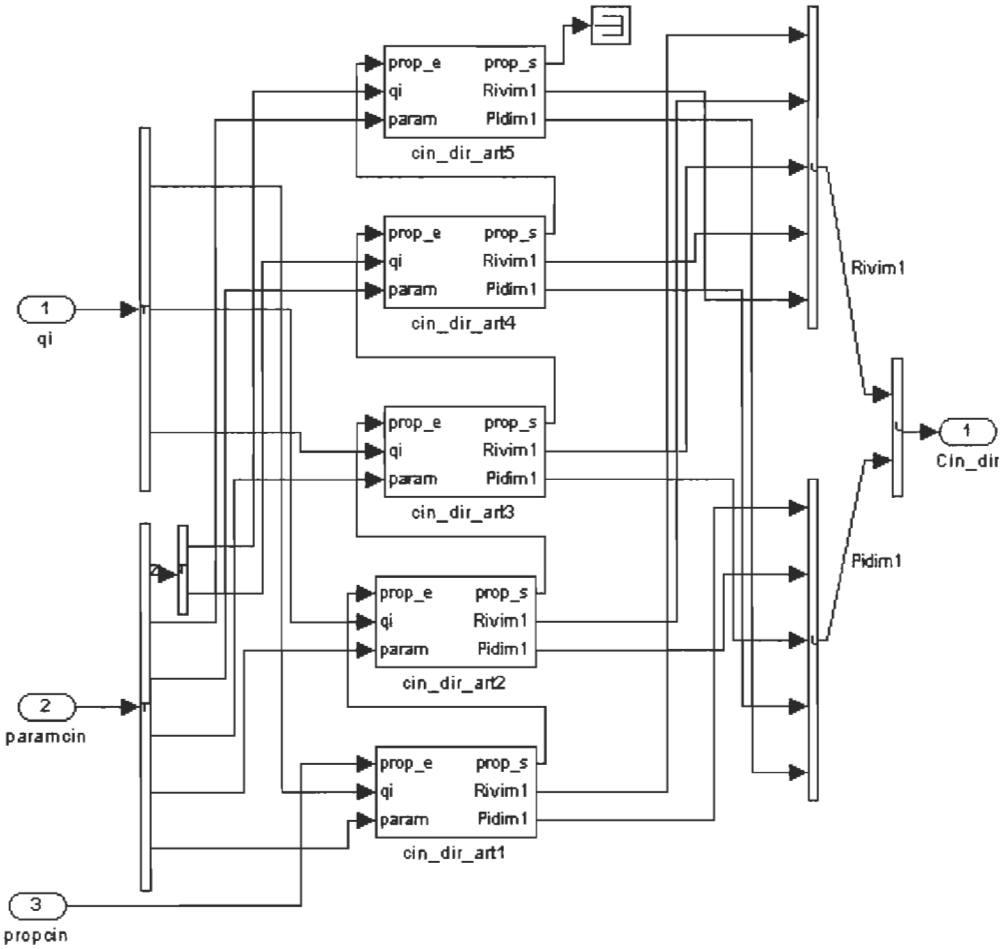

Figure 5-28 : Schéma Simulink™ du bloc de calcul de cinématique directe du doigt

contient les masses, la position des masses, les tenseurs d'inertie et la cinématique directe assiciés au robot. Le vecteur « propcin » contient l'orientation et la position de la base du robot par rapport à une base générale. Le vecteur « paramcin » contient les paramètres Denavit-Hartenberg modifiés définissant la géométrie du robot.

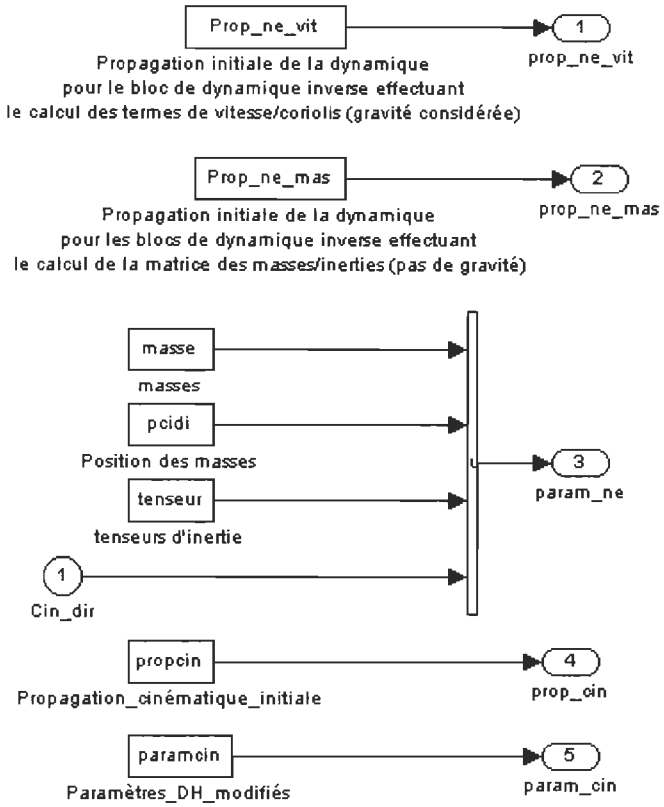

Figure 5-29 : Schéma Simulink™ du bloc des entrées

La figure 5.30 montre l'intérieur du bloc de dynamique directe (Dyndir_modèle). Le bloc « Dyn_inv4 » calcule le vecteur des termes centrifuges, de Coriolis et de gravité tandis que les blocs « Dyn_inv1 » à « Dyn_inv3 » calculent les vecteurs de colonne de la matrice des masses. Ces blocs sont générés automatiquement sous forme de « s-function » en langage C. Le générateur de code utilise les équations (2.8) à (2.22) pour construire les fonctions de calcul de la dynamique inverse articulaire du robot. Le bloc « ResEq » appelle la fonction « divmat.dll » effectuant la résolution d'équation selon la méthode de décomposition LU [29] pour obtenir l'accélération articulaire « qipp ». L'algorithme de résolution d'équation tient compte de la symétrie de la matrice des

masses/inerties pour réduire le nombre d'opérations. Cette fonction est modifiée automatiquement en fonction du nombre d'articulations actives du robot. L'entrée « F/C_ext » contient les forces et couples externes exercés par l'effecteur sur l'environnement. L'entrée « Force/Couple » est l'entrée du signal de commande (forces/couples appliqués à l'articulation). Les entrées « Prop_NE_VG » et « Prop_NE_Masse » contiennent les vitesses et accélérations de la base à utiliser pour le calcul des termes centrifuges, de Coriolis et de gravité (Prop_NE_VG) et pour le calcul des éléments de la matrice des masses (Prop_NE_Masse). L'entrée « Param_NE » contient la cinématique directe ainsi que les paramètres des masses du doigt. Les sorties sont la position « qi », vitesse « qip » et accélération « qipp » articulaires du doigt.

La figure 5.31 montre le schéma d'un bloc de dynamique inverse (la numérotation correspond à celui du calcul de la première colonne de la matrice des masses, le bloc « Dyn_inv1 »). Les entrées sont les forces et couples externes « f_n », la vitesse et accélération articulaires « qid » et « qidd », la vitesse et accélération de la base « Prop_NE » et la cinématique directe et les paramètres des masses « Param_NE ». La sortie est un couple représentant soit une colonne de la matrice des masses, soit le vecteur des termes centrifuges, de Coriolis et de gravité.

Le bloc de recombinaison sert à restructurer les entrées pour les blocs de dynamique inverse. Ce bloc est généré automatiquement sous forme de « s-fonction » en langage C, en fonction du nombre d'articulations.

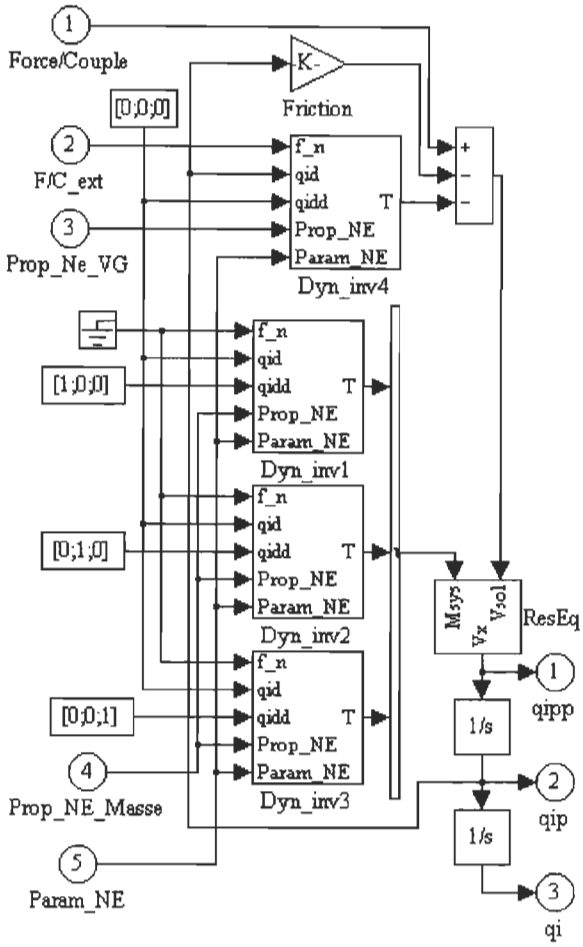

Figure 5-30 : Schéma Simulink™ du modèle de la dynamique directe du doigt

Chacun des blocs « Dyn_inv_art » calcule la dynamique inverse d'une articulation par la méthode récursive de Newton-Euler. Il y a autant de blocs de dynamique inverse qu'il y a d'articulations actives (contrôlées par un moteur ou actionnées par un élément actif quelconque).

La figure 5.32 présente l'intérieur du bloc de dynamique inverse articulaire du doigt. La fonction « dyninvbae » calcule la première itération (base à effecteur) de la dynamique inverse d'une articulation du doigt. La seconde itération (effecteur à base) est calculée

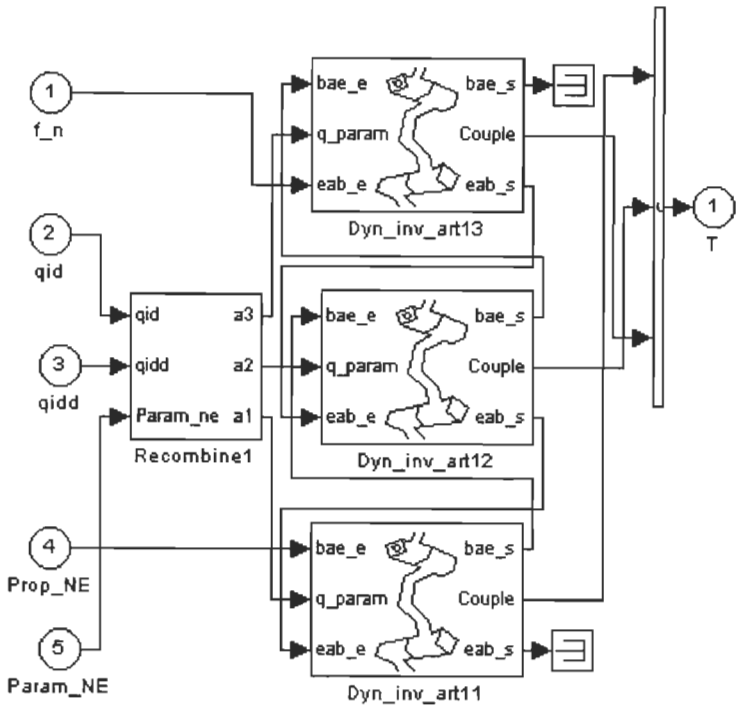

Figure 5-31 : Dynamique inverse du doigt

par la fonction « dyninveab ». Les entrées sont la vitesse, accélération et force/couple d'inertie « bae_e », la cinématique directe, la vitesse et accélération articulaire et les paramètres des masses du membre « q_param » et le vecteur de force/couple externe « eab_e ». Les sorties sont la vitesse, accélération et force/couple d'inertie « bae_s » (qui sera « bae_e » pour l'articulation suivante), le couple obtenu à cette articulation « Couple » et le vecteur de force/couple externe « eab_s » (qui sera « eab_e » pour l'articulation suivante).

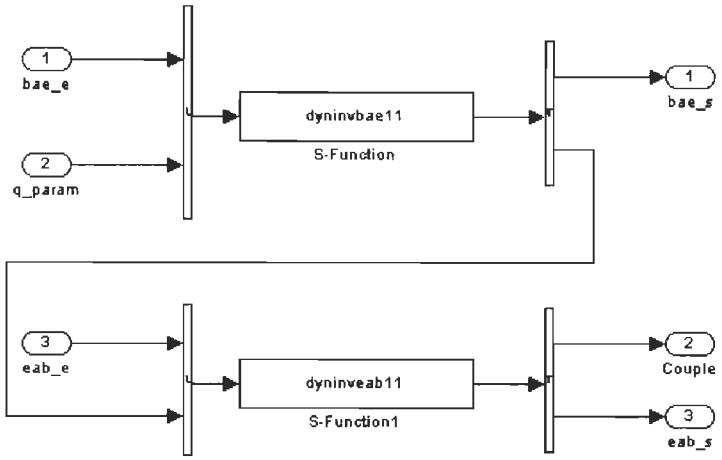

Figure 5-32 : Schéma Simulink™ d'un bloc de dynamique inverse articulaire

5.2.3. Description des essais effectués

Nous utilisons *Simulink*™ de *Matlab*® pour simuler la main robotique. Nous simulons le modèle avec un pas de calcul fixe de 1ms avec l'algorithme Runge-Kutta ODE de 4^{e} ordre. Nous désirons tester le comportement de la main en présence d'un environnement changeant. Les paramètres variants sont la rigidité de l'objet à saisir et l'erreur cinématique du système robot/objet. Une erreur cinématique positive signifie que la distance réelle entre l'effecteur et l'objet est supérieure à la distance connue. La valeur nominale de rigidité pour laquelle le contrôleur de force a été ajusté est k_e=1kN/m . Nous varions les deux paramètres simultanément d'un facteur de 2. Nous ne pouvons pas trop augmenter la rigidité de l'environnement sans ajuster à la baisse le pas de calcul du simulateur. Nous essayons aussi de saisir l'objet avec une rigidité différente pour chaque surface (essai 13). Nous effectuons aussi deux essais avec conditions extrêmes ; nous varions la rigidité de l'objet d'un facteur de 10 tout en gardant une faible erreur

cinématique. Notons que l'objet saisi est centré par rapport aux trois doigts. Les tâches à effectuer sont les suivantes :

- Identification des masses des doigts par une trajectoire aller-retour des doigts ;
- Approche des doigts jusqu'à une certaine distance de la surface à saisir (peut être positive ou négative ; une distance négative signifie que l'effecteur est entré en contact avec la surface avant de tomber en contrôle de force) ;
- Saisie de l'objet par la main et application d'une force de 2N par les doigts (2% de la consigne de force) ;
- Application de la force de consigne de 100N ;
- Relâchement de la force de consigne jusqu'à 2% de sa valeur (2N) ;
- Relâche complète de l'objet (en commande de force) ;
- Retour des doigts à leur position initiale en commande de position.

Le tableau 5.2 montre les paramètres des essais effectués. Les trajectoires de position et de force sont présentées aux figure 5.33 et 5.34. Les conditions initiales de tous les filtres sont mises à 0. Les paramètres de l'identificateur sont les suivants : $\mathbf{P}_0=\mathbf{I}$, $\Delta=100\mathbf{I}$, $\lambda=0.2$, $\rho_0=0.5$, $k_{\lim}=1000$, $\mathbf{R}=500\mathbf{I}$. Les paramètres du contrôleur de force sont : $k_p=0.3$, $k_0=0.01$, $k_1=10$, $k_2=10$.

Tableau 5-2 : Description des essais effectués

Essai	Rigidité de l'objet k_e (N/m)	Erreur cinématique e_c (m)
1	500	-0.02
2	500	-0.005
3	500	0.005
4	500	0.02
5	1000	-0.02
6	1000	-0.005
7	1000	0.005
8	1000	0.02
9	2000	-0.02
10	2000	-0.005
11	2000	0.005
12	2000	0.02
13	1000,500,2000	0.005
14	100	0.005
15	10000	0.005

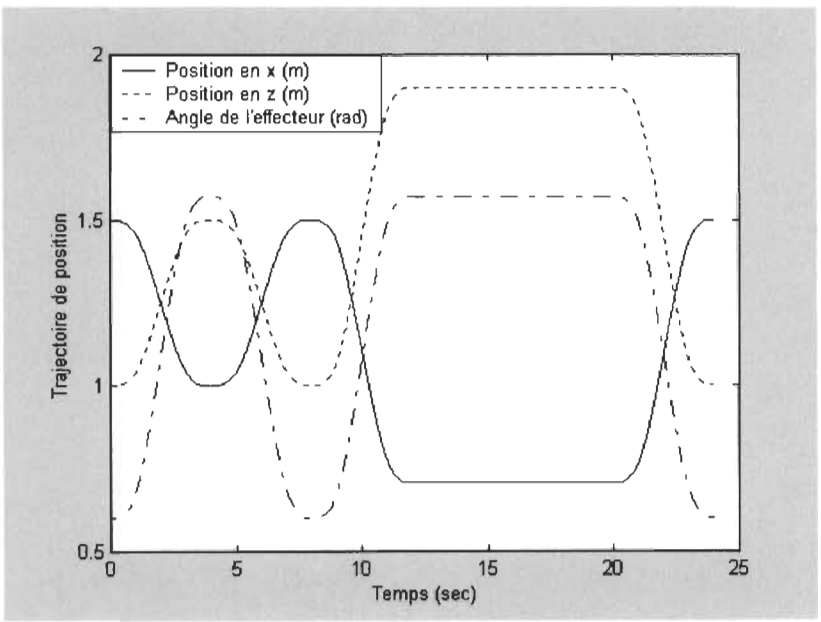

Figure 5-33 : Trajectoire de position articulaire

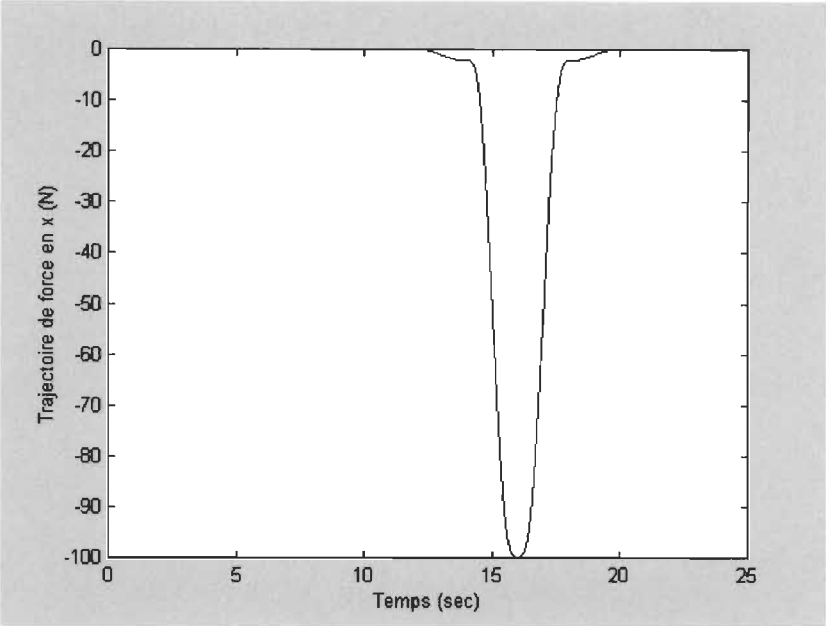

Figure 5-34 : Trajectoire de force en x (domaine galliléen)

5.2.4. Résultats de simulation

Les résultats obtenus à la suite des essais décrits à la section précédente sont tous caractérisés par la même forme. La figure 5.35 montre les résultats d'identification des masses pour le cas nominal (essai 7 du tableau 5.2) i.e. les valeurs de rigidité de l'objet et d'erreur cinématique pour lesquelles le contrôleur de force a été ajusté. Nous montrons seulement les résultats du doigt 1 car les résultats des deux autres doigts sont identiques. Nous observons que l'identificateur réussi à estimer les masses du doigt en deux secondes. Par la suite, l'erreur d'estimation est toujours inférieure à 5%, ce qui est amplement suffisant pour commander efficacement le manipulateur.

La figure 5.36 montre la forme de l'erreur de force, de l'erreur de position en x et en z ainsi que de l'erreur d'orientation de l'effecteur obtenues durant le premier essai pour le

doigt 1 (rigidité de 500N/m et erreur cinématique de -20mm). Les résultats pour les doigts 2 et 3 sont identiques à cause de la symétrie.

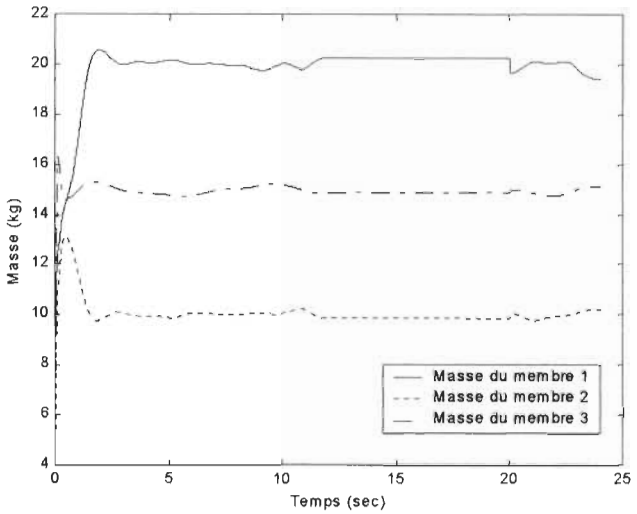

Figure 5-35 : Masses identifiées pour le cas nominal (rigidité de 1kN/m et erreur cinématique de 5mm)

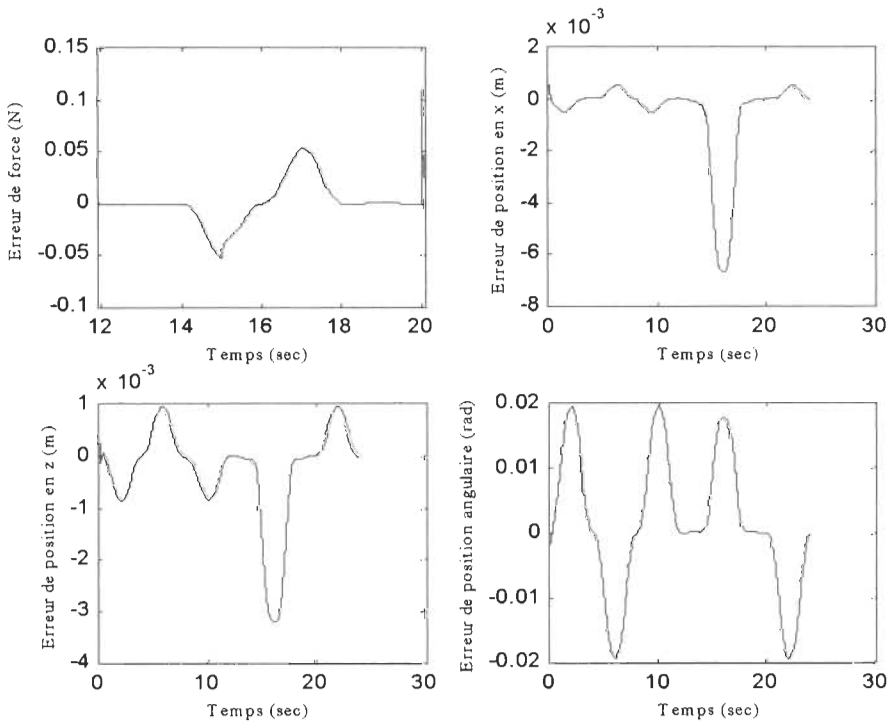

Figure 5-36 : Résultats obtenus pour l'essai 1 (rigidité de l'objet de 500N/m et erreur cinématique de -20mm)

Nous remarquons qu'il y a un pic d'erreur (0.1% d'erreur par rapport à la consigne de 100N) lors de la relâche du contact entre le doigt et l'objet. Cette erreur est causée par la remise à zéro de l'intégrateur du contrôleur de force au début de l'éloignement du doigt, après la tâche de contact. Ailleurs, l'erreur de force reste sous 0.05% de la consigne. Le pic d'erreur de force est moins important lors de l'essai 2 (résultats à l'annexe G) car les doigts exercent une pression moins forte sur l'objet lorsque le contrôleur de force entre en action. L'erreur de position en x obtenue durant le contact (de t=12s à t=20s) importe peu car elle dépend de la rigidité du contrôleur de position. L'erreur de position en x reste sous 7mm. L'erreur de position en z reste inférieure à 3mm pendant le contact. Considérant la longueur des membres, l'erreur de 3mm est faible, ce qui montre que le contrôleur est directionnel, c'est-à-dire qu'il est possible de commander une direction où il y a une contrainte sans déranger la commande dans les autres directions. L'erreur d'orientation maximale de l'effecteur est 1.2° ; le contact n'affecte pas beaucoup l'orientation de l'effecteur. Finalement, nous observons que la phase d'apprentissage initiale des masses cause une petite erreur de position se corrigeant en moins de 1 seconde. Ce régime transitoire survient parce que les conditions initiales des filtres de l'identificateur sont nulles et, pour les deux premières secondes de simulation, l'identificateur est en train de converger. Les valeurs connues des masses sont donc très différentes des valeurs réelles. Il est nécessaire de conserver les valeurs initiales des filtres à 0 pour des raisons de robustesse. En effet, nous avons cherché les conditions initiales idéales. Pour ce faire, nous avons effectué des simulations où les doigts effectuent des trajectoires de position aller-retour sans contact de façon à obtenir le

régime permanent des états des filtres de l'identificateur. Nous avons ensuite utilisé la valeur des états des filtres en régime permanent comme conditions initiales pour les simulations. Nous avons obtenu d'excellents résultats lorsque les masses réelles des doigts avaient la même valeur que lorsque nous avions simulé le régime permanent. La convergence était complète en 0.1 seconde (au lieu de 2 secondes pour la simulation avec conditions initiales nulles). Par contre, lorsque les masses réelles étaient différentes (d'un facteur de 2 par exemple) des masses utilisées pour simuler le régime permanent, alors la convergence était dix fois plus lente que pour une simulation avec conditions initiales nulles.

Lors du troisième essai (k_e=$500N/m$ et e_c=$5mm$), nous obtenons les erreurs montrées à la figure 5.37. Les erreurs de position sont sensiblement les mêmes que pour le premier essai. Il apparaît une oscillation amortie dans l'erreur de force lors du contact. Cette perturbation se produit parce que le contrôleur de force entre en action avant que l'effecteur entre en contact avec la surface de l'objet. L'effecteur arrive donc sur la surface à une vitesse plus grande que prévue. Cette observation est plus évidente avec les résultats de l'essai 4 (erreur cinématique de 20mm) qui sont présentés à l'annexe G.

Le cinquième essai consiste à répéter l'essai 1 mais avec une rigidité de surface de l'objet de 1kN/m (rigidité nominale). Les erreurs de position et de force obtenues suite à cet essai sont présentées à la figure 5.38. Les erreurs de position en x, en z et en rotation sont légèrement plus grandes de 5% à 10% par rapport à l'essai 1. Par contre, l'erreur de force est 48% plus faible que pour l'essai 1.

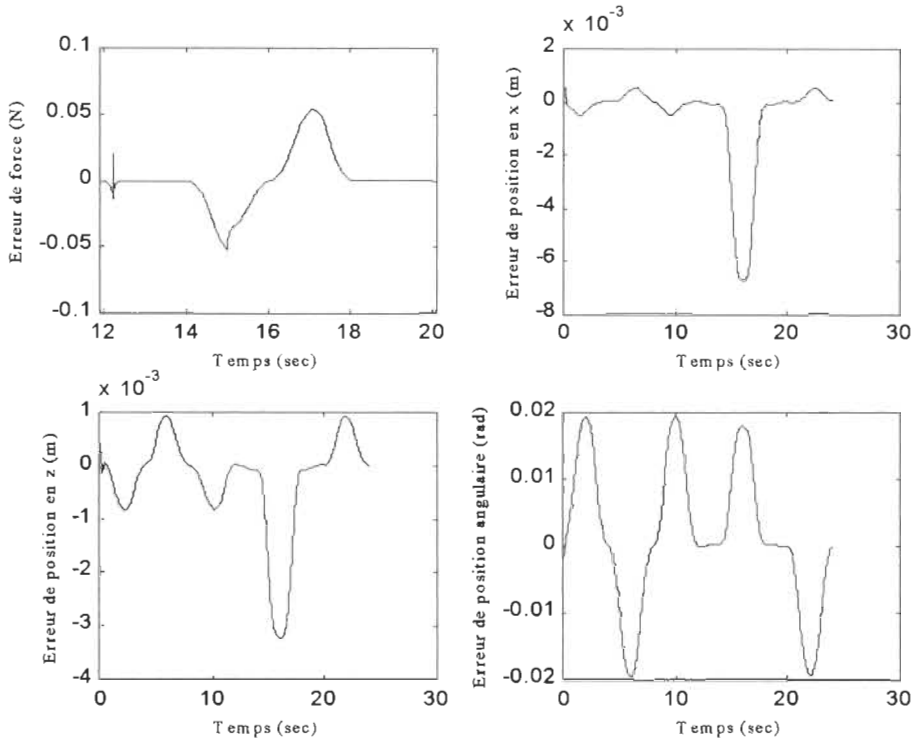

Figure 5-37 : Résultats obtenus lors du troisième essai (rigidité de l'objet de 500N/m et erreur cinématique de 5mm)

Nous avons comparé les résultats de l'essai 6 (présentés à l'annexe G) avec ceux de l'essai 2 et nous sommes arrivés aux mêmes conclusions qu'en comparant les résultats de l'essai 5 avec ceux de l'essai 1.

La figure 5.39 montre les erreurs obtenues suite au septième essai ; il s'agit de l'essai nominal (l'essai pour lequel les contrôleurs ont été ajustés). La rigidité est de 1kN/m et l'erreur cinématique est de 5mm. Les erreurs de position sont légèrement plus grandes que durant l'essai 3. L'erreur de force moyenne est un peu plus faible que pour l'essai 3 mais le régime transitoire d'erreur de force est un peu plus grand lors du contact (à

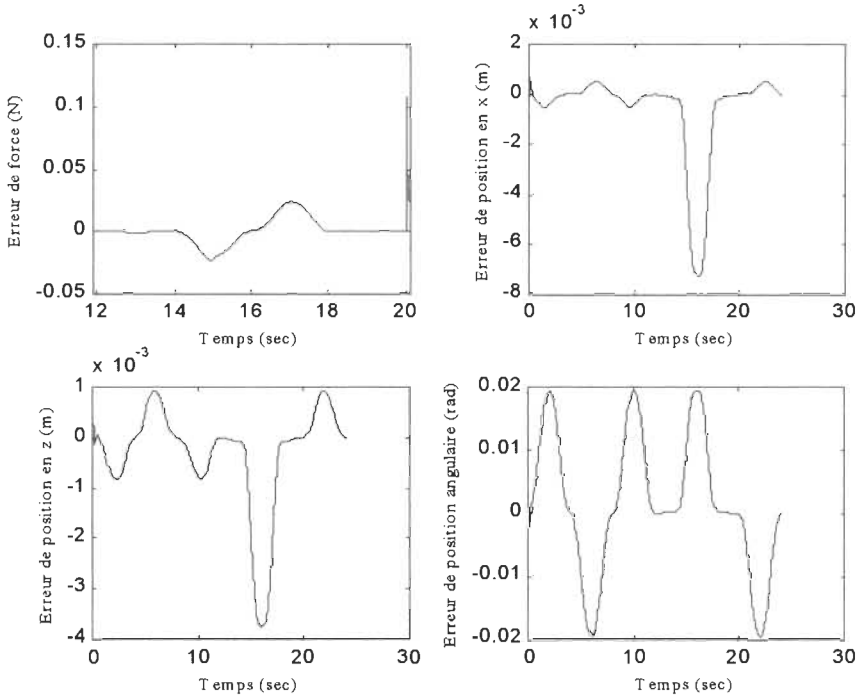

Figure 5-38 : Résultats obtenus suite à l'essai 5 (rigidité de l'objet de 1kN/m et erreur cinématique de –20mm)

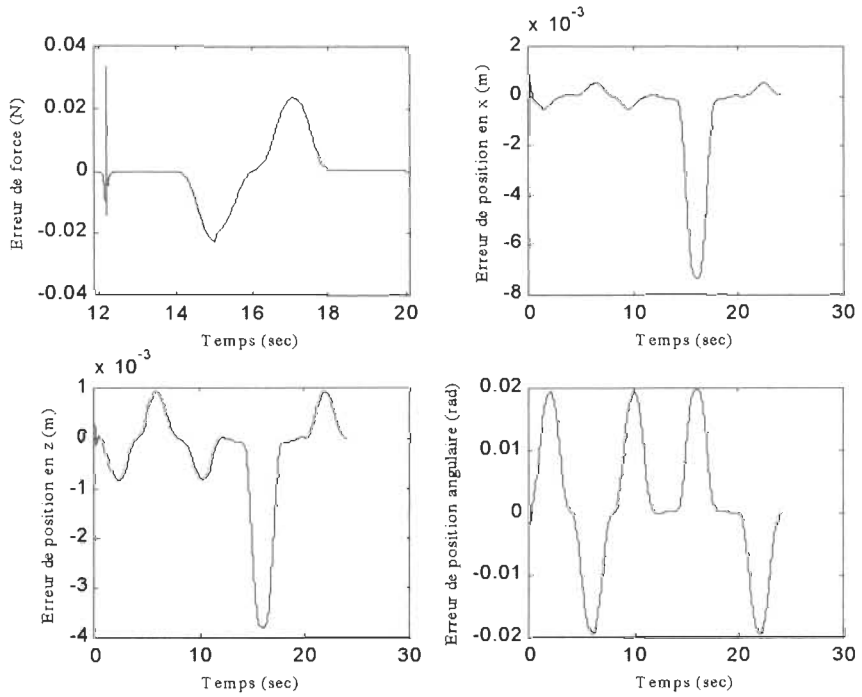

Figure 5-39 : Résultats obtenus pour l'essai 7 (rigidité de l'objet de 1kN/m et erreur cinématique de 5mm)

t=12s). Nous pouvons poser les mêmes remarques en comparant les résultats de l'essai 8 (présentés à l'annexe G) avec ceux de l'essai 4.

Nous avons ensuite comparé les résultats obtenus lors des essais 9 à 12 avec ceux des essais 5 à 8. La variation de la rigidité de l'objet de 1kN/m à 2kN/m a le même effet sur les résultats que pour la variation de 500N/m à 1kN/m. Les résultats de ces essais sont présentés à l'annexe G.

Nous avons effectué un essai pour lequel la rigidité des surfaces de l'objet est différente pour chacune des surfaces. La surface contactée par le doigt 1 (pour lequel nous observons les erreurs de forces et de position) est caractérisée par une rigidité de 1kN/m. Les deux autres surfaces ont une rigidité de 500N/m et 2kN/m respectivement pour les doigts 2 et 3. L'erreur cinématique est de 5mm. Les résultats pour le doigt 1 sont présentés à la figure 5.40. Nous constatons que les résultats sont pratiquement identiques à ceux de l'essai 7 (rigidité de 1kN/m et erreur cinématique de 5mm) même si l'interaction entre les doigts est différente, ce qui prouve une certaine robustesse face aux perturbations externes de position. La seule différence marquée est le pic d'erreur de force présent lors de la relâche du contact. Cette erreur est causée par le fait que l'objet se déplace pendant le contact car les doigts réagissent de façon différente à cause de la différence de rigidité des surfaces. Les résultats pour les doigts 2 et 3 sont présentés aux figures 5.41 et 5.42. Les résultats obtenus sont pratiquement identiques aux résultats obtenus lors des essais 3 et 11 pour les doigts 2 et 3 respectivement. La seule différence visible se trouve au niveau de l'erreur de force du doigt contactant la surface dont la

rigidité est de 2kN/m. Lors de l'essai 11 (rigidité de 2kN/m pour les 3 doigts), l'erreur de force était nulle lors de la relâche de l'objet. Par contre, lors de l'essai 13 (rigidité différente pour les trois doigts), l'objet bouge pendant le contact et l'erreur cinématique de +5mm devient négative pour le doigt 3, ce qui a comme effet de causer un second contact lors de la remise à zéro de l'intégrateur du contrôleur de force après la relâche de l'objet. La figure 5.43 présente la variation des coordonnées des surfaces de l'objet vues par les trois doigts. Les référentiels utilisés sont ceux des doigts ; les coordonnées en x et en y de la surface de contact pour le doigt x sont données par rapport au référentiel du doigt x.

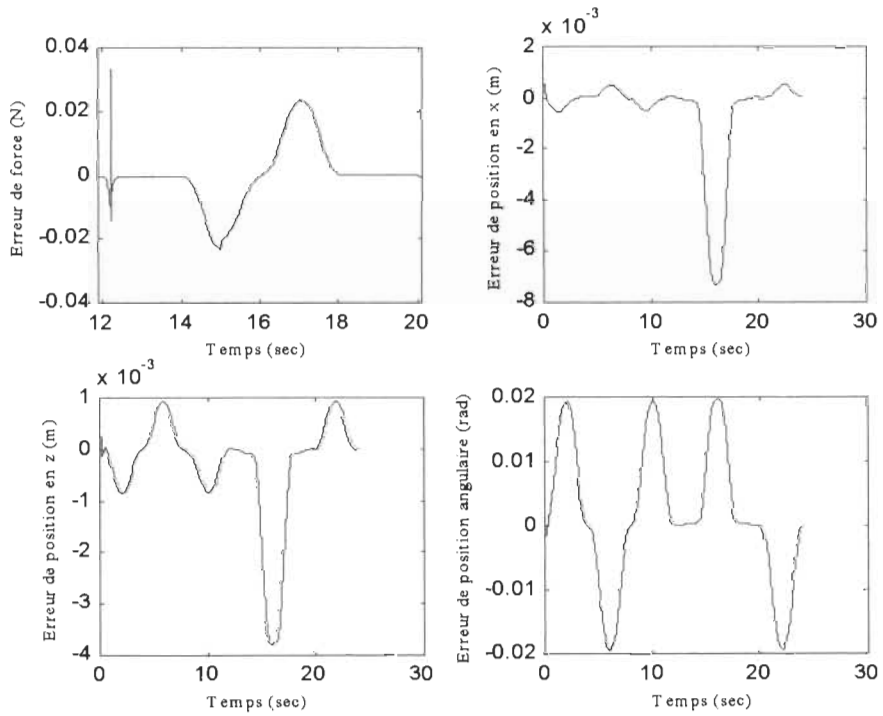

Figure 5-40 : Résultats pour le doigt 1 de l'essai avec une rigidité de 1kN/m, 500N/m et 2kN/m pour les doigts 1, 2 et 3 respectivement et pour une erreur cinématique de 5mm

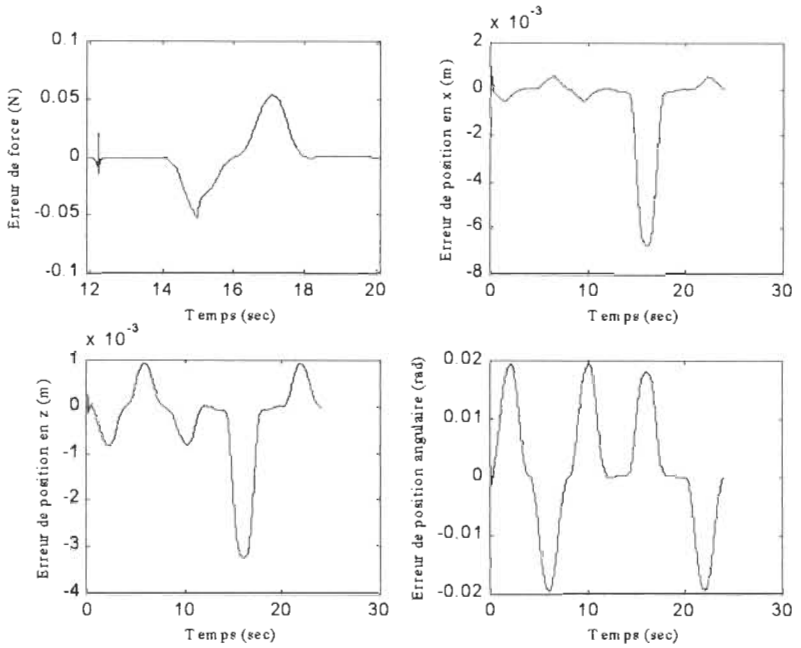

Figure 5-41 : Résultats pour le doigt 2 de l'essai avec une rigidité de 1kN/m, 500N/m et 2kN/m pour les doigts 1, 2 et 3 respectivement et pour une erreur cinématique de 5mm

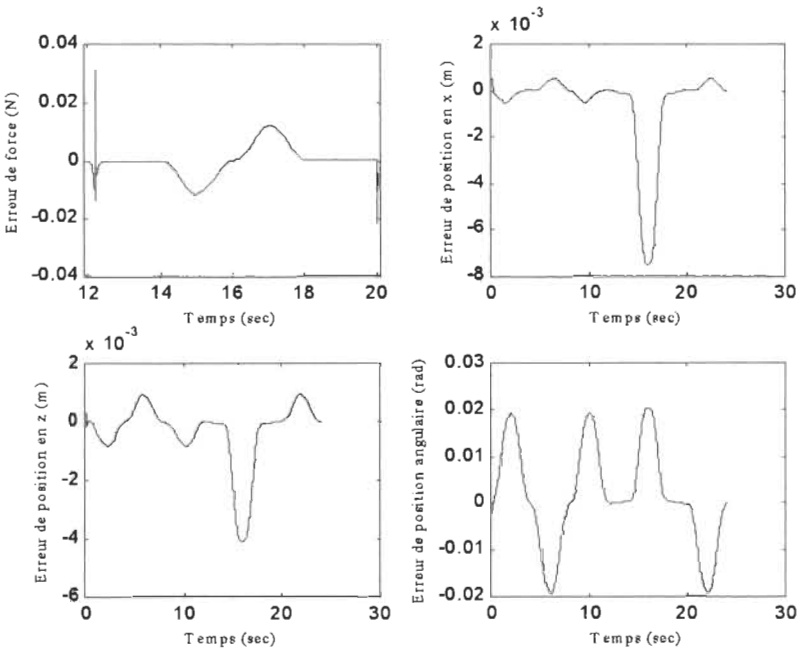

Figure 5-42 : Résultats pour le doigt 3 de l'essai avec une rigidité de 1kN/m, 500N/m et 2kN/m pour les doigts 1, 2 et 3 respectivement et pour une erreur cinématique de 5mm

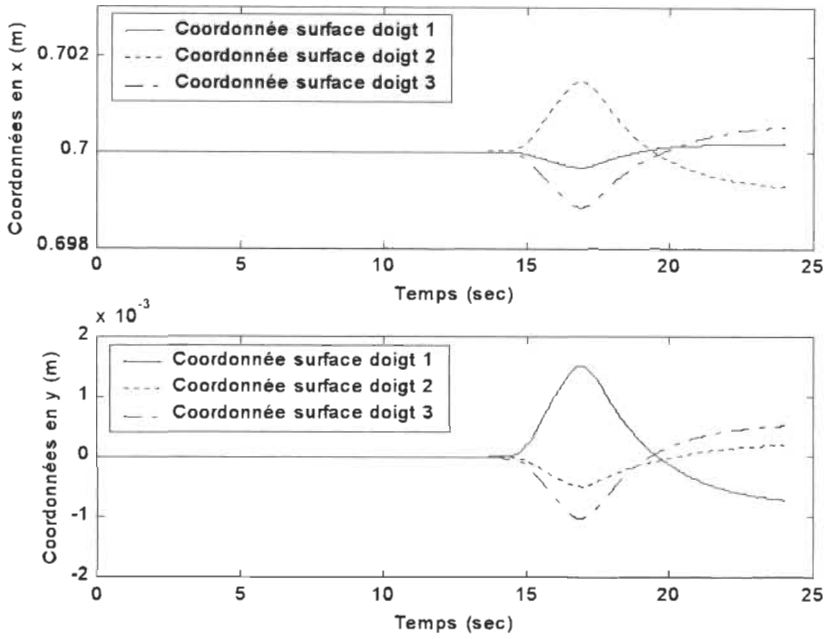

Figure 5-43 : Coordonnées relatives des surfaces de l'objet

Finalement, nous avons effectué deux essais en variant d'un facteur de 10 la rigidité de l'objet. L'erreur cinématique est posée à 5mm. Le premier essai extrême utilise un objet dont la rigidité de ses surfaces est de 100N/m. Nous posons la consigne de force à 10N au lieu de 100N pour que l'effecteur ne parcoure pas une trop grande distance durant le contact. Les résultats de simulation sont montrés à la figure 5.43. L'erreur de force maximale est de 0.2% de la consigne. Les erreurs de position maximales en x et en z sont respectivement 0.5mm et 1mm tandis que l'erreur maximale d'orientation de l'effecteur est de 1.2°. Les performances sont donc bonnes lorsque nous diminuons grandement la rigidité de l'objet.

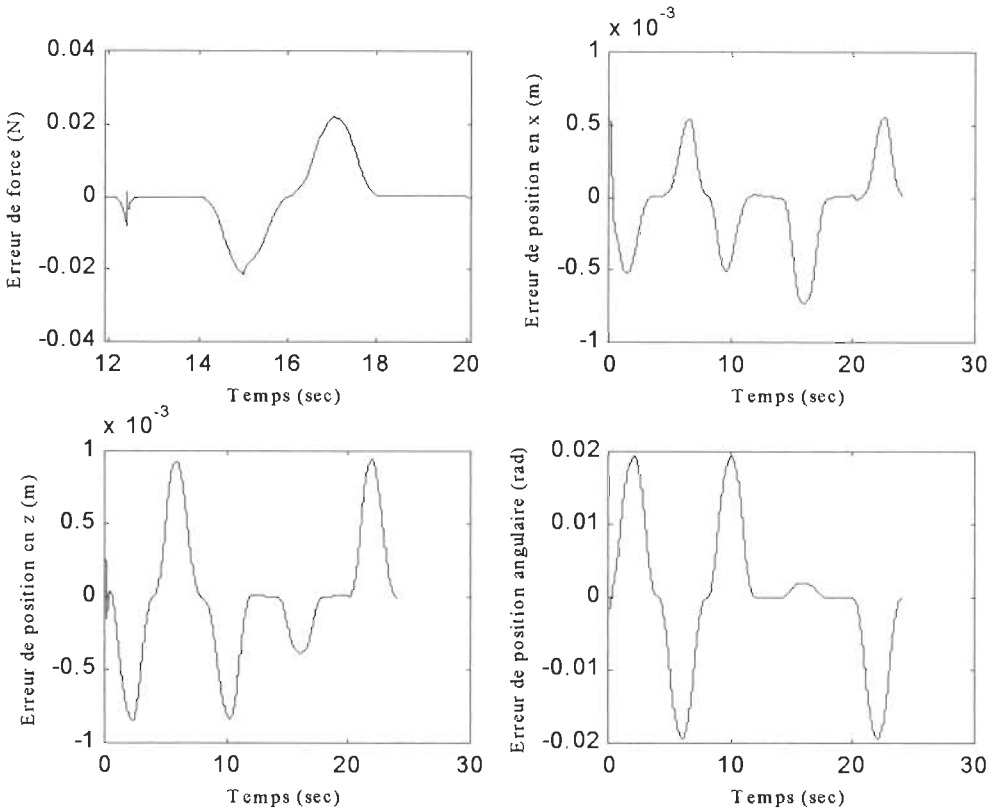

Figure 5-44 : Résultats de l'essai avec un objet dont la rigidité est de 100N/m et erreur cinématique de 5mm

Le deuxième essai avec conditions extrêmes est effectué en utilisant un objet dont la rigidité est de 10kN/m (dix fois plus dur que l'objet nominal). La consigne de force est de 100N et l'erreur cinématique est de 5mm. Pour pouvoir simuler le système dans ces conditions, nous devons soit diminuer le pas de calcul d'un facteur de 10 ou soit simuler avec un pas de calcul variable (avec Runge-Kutta ODE45). Nous avons adopté la dernière solution. Normalement, nous simulons avec un pas fixe car c'est un pré-requis pour la simulation en temps réel avec Real-Time Workshop mais il est impossible de simuler le système en temps réel dans ces conditions car le pas de calcul nécessaire est trop petit. La simulation à pas variable est plus rapide à exécuter car le pas diminue

seulement quand c'est nécessaire pour obtenir une précision acceptable. Les résultats de l'essai avec rigidité de 10kN/m sont montrés à la figure 5.44. Nous observons que le pic d'erreur de force obtenu lorsque le doigt 1 entre en contact avec l'objet est 5 à 10 fois plus grand que pour les essais précédents. Notons toutefois que l'erreur maximale est faible (0.15%). L'augmentation de l'erreur de force est causée par l'éloignement des conditions de simulation du point d'opération pour lequel le contrôleur de force a été ajusté.

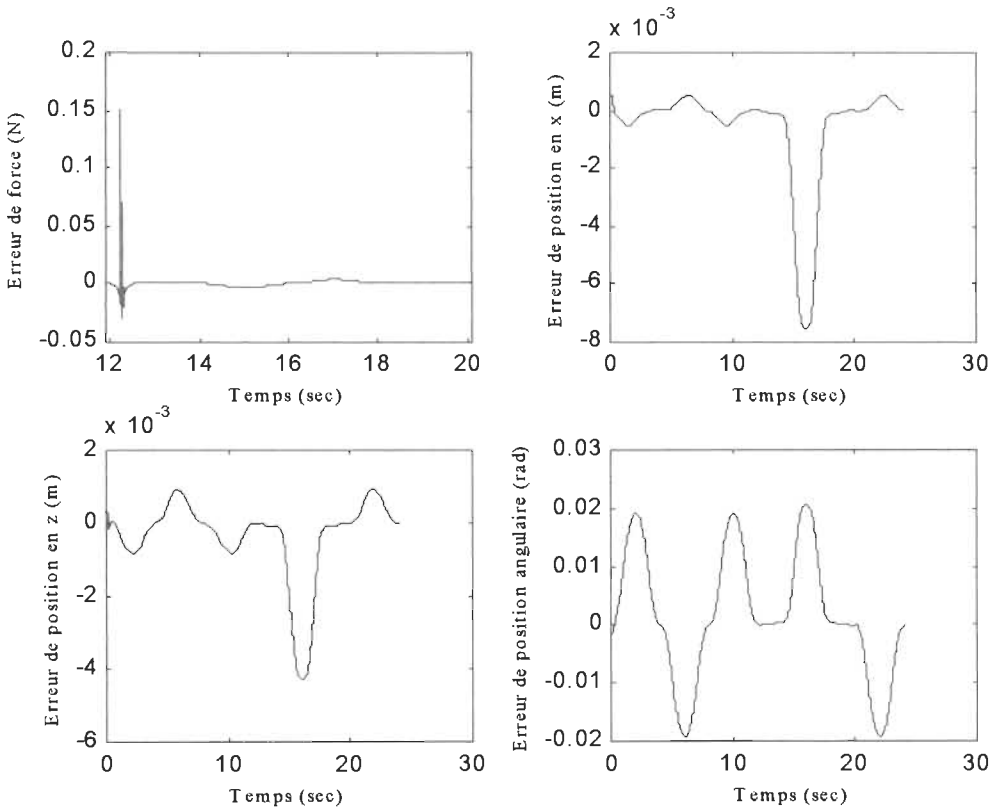

Figure 5-45 : Résultats de l'essai avec conditions extrême utilisant un objet dont la rigidité est de 10kN/m et erreur cinématique de 5mm

Pour comparer quantitativement les résultats des essais entre eux, nous avons utilisé l'erreur absolue moyenne et l'erreur absolue maximale. L'erreur absolue moyenne est la

moyenne de la valeur absolue des erreurs. Les erreurs observées sont l'erreur absolue moyenne en force e_{moyF}, l'erreur absolue maximale en force $e_{\max F}$, l'erreur absolue moyenne de position en x e_{moyPx}, l'erreur absolue maximale de position en x $e_{\max Px}$, l'erreur absolue moyenne de position en z e_{moyPz}, l'erreur absolue maximale de position en z $e_{\max Pz}$, l'erreur absolue moyenne d'orientation de l'effecteur $e_{moy\theta}$ et l'erreur absolue maximale d'orientation de l'effecteur $e_{\max \theta}$. Le tableau 5.3 résume les résultats obtenus lors des 13 premiers essais où la rigidité k_e de l'objet et l'erreur cinématique e_c variaient. Les résultats sont classés en ordre croissant de performance avec comme critère l'erreur absolue moyenne de force.

Tableau 5-3 : Résumé des résultats obtenus pour les essais avec rigidité de l'objet et erreur cinématique variables classés en ordre décroissant d'erreur absolue moyenne de force

Essai	k_e	e_c	e_{moyF}	$e_{\max F}$	e_{moyPx}	$e_{\max Px}$	e_{moyPz}	$e_{\max Pz}$	$e_{moy\theta}$	$e_{\max \theta}$
1	500	-0.02	4.25E-03	1.09E-01	7.45E-04	6.69E-03	5.79E-04	3.20E-03	8.07E-03	1.94E-02
4	500	0.02	4.09E-03	5.34E-02	7.58E-04	6.75E-03	5.79E-04	3.23E-03	8.09E-03	1.95E-02
2	500	-0.005	3.98E-03	5.34E-02	7.49E-04	6.73E-03	5.81E-04	3.22E-03	8.08E-03	1.94E-02
3	500	0.005	3.94E-03	5.34E-02	7.52E-04	6.76E-03	5.81E-04	3.23E-03	8.09E-03	1.94E-02
5	1000	-0.02	2.22E-03	1.08E-01	7.79E-04	7.25E-03	6.17E-04	3.76E-03	8.18E-03	1.95E-02
8	1000	0.02	2.07E-03	8.27E-02	7.92E-04	7.32E-03	6.17E-04	3.80E-03	8.20E-03	1.97E-02
6	1000	-0.005	1.95E-03	2.69E-02	7.83E-04	7.30E-03	6.19E-04	3.79E-03	8.19E-03	1.97E-02
13 (doigt1)	multiple	0.005	1.93E-03	3.33E-02	7.85E-04	7.32E-03	6.19E-04	3.80E-03	8.20E-03	1.98E-02
7	1000	0.005	1.92E-03	3.33E-02	7.86E-04	7.32E-03	6.19E-04	3.80E-03	8.20E-03	1.98E-02
9	2000	-0.02	1.33E-03	8.76E-02	7.90E-04	7.45E-03	6.35E-04	4.05E-03	8.22E-03	2.02E-02
12	2000	0.02	1.18E-03	5.58E-02	8.02E-04	7.52E-03	6.35E-04	4.09E-03	8.23E-03	2.04E-02
10	2000	-0.005	1.06E-03	2.19E-02	7.93E-04	7.50E-03	6.37E-04	4.08E-03	8.23E-03	2.03E-02
11	2000	0.005	1.03E-03	3.09E-02	7.96E-04	7.52E-03	6.37E-04	4.09E-03	8.23E-03	2.04E-02

La figure 5.46 montre les tendances générales des erreurs de force et de position en fonction de l'erreur cinématique et de la rigidité de l'objet. Les tendances sont évaluées en fonction de l'essai nominal, i.e. erreur cinématique de 5mm et rigidité de l'objet de 1kN/m. Lorsque nous varions la rigidité de l'objet de 500N/m à 2kN/m, l'erreur cinématique est de 5mm. Lorsque nous varions l'erreur cinématique de –20mm à 20mm, la rigidité est de 1kN/m. Nous observons que le facteur dominant affectant l'erreur de force absolue moyenne est la rigidité de l'objet ; l'erreur de force absolue moyenne est inversement proportionnelle à la rigidité de l'objet. Ensuite, plus la valeur absolue de l'erreur cinématique augmente, plus l'erreur de force absolue moyenne et maximale augmentent. Par contre, l'erreur de force est peu sensible à la variation de l'erreur cinématique ; une grande augmentation de l'erreur cinématique (400%) cause une petite augmentation de l'erreur de force (14%). Ensuite, une variation de la rigidité de l'objet cause une petite augmentation de l'erreur de position ; une augmentation de 400% de la rigidité de l'objet cause une augmentation de l'erreur de position de 2 à 9%. L'erreur de position augmente parce que la rigidité du contrôleur de position est constante tandis que la rigidité de l'objet augmente. Il est donc plus contraignant pour le contrôleur de position d'appliquer les couples articulaires nécessaires à la convergence vers la trajectoire désirée. L'erreur de position dans les directions non contraintes (position en z et orientation de l'effecteur) est pratiquement insensible aux variations de l'erreur cinématique. Par contre, l'erreur de position dans la direction contrainte (position en x) augmente lorsque l'erreur cinématique augmente. Cette augmentation est très faible (environ 1% d'augmentation de l'erreur de position pour une variation de l'erreur cinématique de 400%).

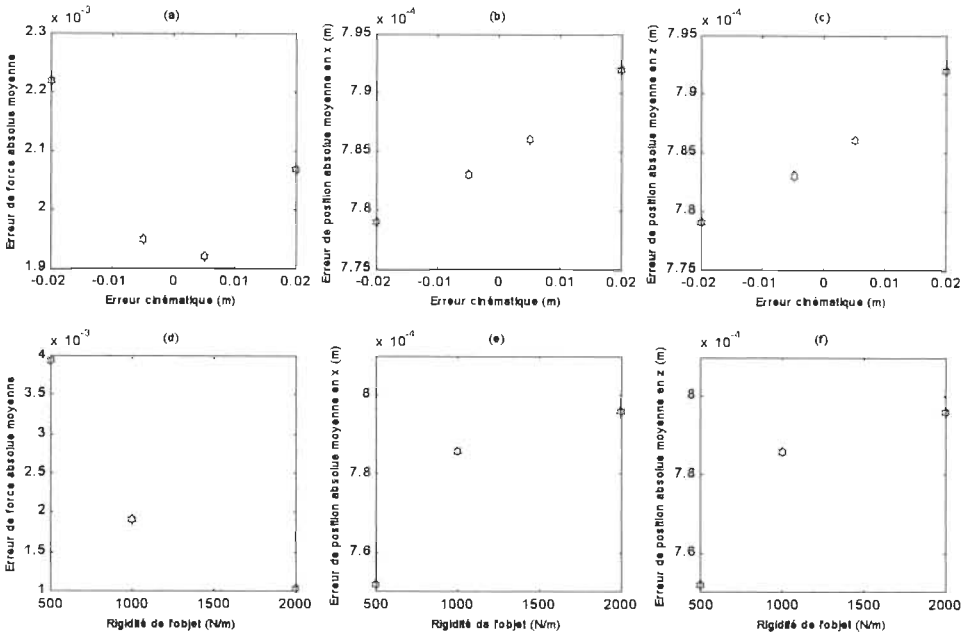

Figure 5-46 : Tendances de l'erreur de position et de force en fonction de l'erreur cinématique et de la rigidité de l'objet

5.2.5. Étude de la variation du temps de calcul en fonction du nombre de nœuds de calcul et de l'algorithme utilisé

Nous avons simulé un doigt de la main sous la formulation de Lagrange et Newton-Euler de façon à connaître l'influence de la formulation sur le temps de calcul dans l'environnement Simulink™. Nous avons enlevé les contrôleurs et l'environnement pour obtenir un temps de calcul dépendant uniquement du calcul de la dynamique directe du doigt. Le doigt démarre à l'horizontale pour ensuite tomber et se balancer. Cette simulation a été effectuée sur un Pentium II 350Mhz avec 192 Mbytes de RAM, sur Windows 95. Le temps de simulation est de 10 secondes. Nous avons utilisé l'algorithme de Runge-Kutta ODE4 à pas fixe de 1 ms. Les simulateurs utilisant la formulation de Lagrange et de Newton-Euler ont terminé respectivement la simulation en

4.89 et 34.3 secondes. La lenteur du modèle Newton-Euler peut être partiellement expliquée par le nombre de blocs *Simulink*™ 12 fois plus élevé par rapport au modèle de Lagrange. Cette différence de nombre de blocs a pour conséquence un plus grand temps utilisé pour le fonctionnement de la structure du programme i.e. le temps de communication entre fonctions (instructions PUSH, PULL etc.). De plus, il est plus facile de simplifier les équations du modèle sous la forme de Lagrange ; le code écrit sous cette forme est donc plus compact. Nous pourrions obtenir de meilleurs résultats en écrivant tous les blocs du modèle Newton-Euler dans le même programme.

Peu importe le modèle utilisé, il est présentement impossible (même avec un Pentium III 700Mhz) de simuler la main robotique en temps réel avec un pas de calcul de 1ms et les modèles employés. Nous recourons à l'utilisation d'un système multi-PC branché en réseau avec des cartes Firewire de 200Mbit/sec (protocole IEEE 1394). Les noeuds de calcul esclaves et le noeud maître sont des Pentium II 233Mhz avec 16Mo de RAM configurés pour QNX. Le noeud console fonctionne avec Windows NT et sert seulement de lien permettant à l'usager de recueillir des informations pendant la simulation. Le lien séparant la console du maître fonctionne en TCP-IP à une vitesse de 10Mbit/sec.

L'efficacité de la simulation en parallèle peut être mesurée par :

$$eff = \frac{t_1}{t_p n_p} \tag{5.36}$$

où t_1 est le temps de calcul en séquentiel, t_p est le temps de calcul en parallèle et n_p est le nombre de processeurs utilisés pour le calcul en parallèle. Nous simulons le modèle de la main formulé selon Lagrange-Euler avec les contrôleurs de position et de force et le planificateur de tâches. Nous commençons par mesurer le temps nécessaire pour effectuer un pas de calcul en séquentiel (simulé sur le noeud maître) et obtenons 2200µs. Nous séparons ensuite le code en 4 parties comme montré à la figure 5.47. Pour séparer le code séquentiel, nous utilisons le séparateur de code de Opal-rt Technologies inc. [24]. Ce séparateur permet de passer d'un schéma Simulink™ séquentiel à autant de schémas qu'il y a de noeuds de calcul. Les schémas Simulink™ générés sont ensuite compilés par Real-Time Workshop. Chaque modèle de doigt est simulé sur un noeud esclave tandis que le modèle de l'objet est simulé sur le noeud maître. Les principales étapes de la simulation sont les suivantes. Premièrement, à chaque pas de calcul, les trois noeuds esclaves et le noeud maître reçoivent les signaux en provenance des autres noeuds. Ensuite, les noeuds calculent leurs sorties. Finalement, les sorties des noeuds sont publiées sur le réseau Firewire. Avec la topologie montrée à la figure 5.47, le temps nécessaire à l'exécution d'un pas de calcul est de 920µs, ce qui correspond à une efficacité de 60%. Ce mauvais rendement est causé par un déséquilibre de la charge de calcul car le modèle de l'objet est beaucoup plus compact que le modèle d'un doigt. Malgré le bas rendement de cette topologie, nous pouvons simuler en temps réel car le temps nécessaire pour effectuer un pas de calcul est inférieur à 1ms (le pas de calcul en temps de simulation).

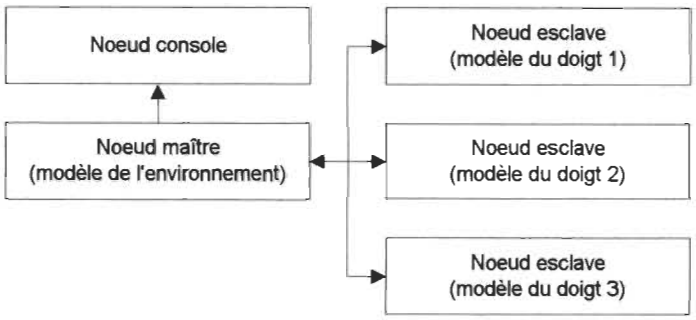

Figure 5-47 : Répartition des tâches pour la simulation en parallèle sur 4 noeuds

L'essai suivant est effectué en séparant le code en 2 parties. La figure 5.48 montre la répartition des tâches sur les trois noeuds. Nous simulons deux des doigts sur un noeud esclave tandis que le noeud maître simule le troisième doigt et l'objet. Le temps de calcul obtenu est de 1580µs, ce qui correspond à une efficacité de 70%. Nous ne pouvons pas faire de temps réel avec une précision suffisante avec cette topologie.

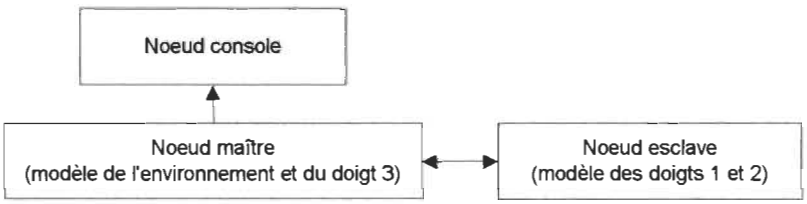

Figure 5-48 : Répartition des tâches pour la simulation en parallèle sur 2 noeuds

Finalement, en séparant le code en 3 parties comme à la figure 5.49 (2 noeuds esclaves simulant 2 doigts et le noeud maître simulant un doigt et l'objet), nous avons obtenu un temps de calcul de 880µs et donc une efficacité de 83%. Nous pouvons utiliser cette topologie en temps réel car le pas de calcul est inférieur à 1ms. De plus, cette topologie est plus rapide que la topologie à quatre noeuds, ce qui prouve qu'il est très important de bien équilibrer les charges sur tous les noeuds de calcul.

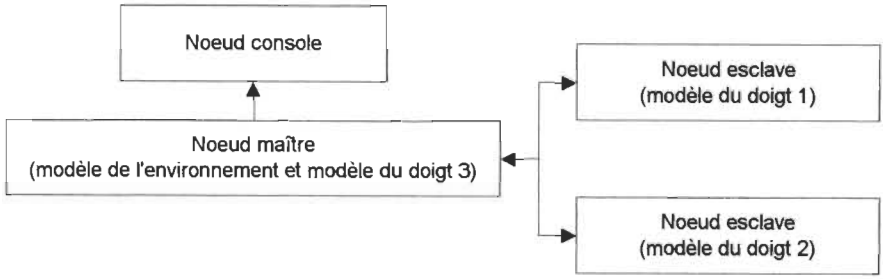

Figure 5-49 : Répartition des tâches pour la simulation en parallèle sur 3 noeuds

Les cartes Firewire utilisées présentement ont un temps minimum réservé à la communication ("overhead") d'environ 50µs. Si nous séparons un algorithme séquentiel à un trop faible grain (beaucoup de noeuds), alors il sera impossible d'atteindre le temps réel à cause de ce temps de communication. Avec les nouvelles cartes Firewire de 400 Mbit/sec, le temps de communication minimum est inférieur.

5.3. Sommaire

Dans ce chapitre, nous avons validé le générateur de code, les lois de commande et la stratégie de saisie présentés dans les chapitres précédents. Le générateur de code nous a permis d'obtenir automatiquement un fort pourcentage du code de simulation du système robotique. Certaines parties du code comme la cinématique inverse et le modèle de l'objet ont été calculés à la main. Il a aussi fallu placer manuellement les blocs générés automatiquement à l'intérieur du modèle du robot conçu. Pour valider la loi de commande position/force, nous avons effectué des essais de saisie d'un objet sous différentes conditions de l'environnement. Nous avons montré que le contrôleur position/force est peu sensible aux variations de l'environnement.

Nous avons conçu un programme de visualisation permettant de voir évoluer le manipulateur dans l'espace sans avoir à interpréter de graphique.

Nous avons ensuite comparé les algorithmes de Lagrange-Euler et de Newton-Euler du point de vue du temps nécessaire pour effectuer un pas de calcul. Nous avons observé que le modèle de la dynamique directe formulé selon Lagrange-Euler est 7 fois plus rapide que le modèle formulé selon Newton-Euler lors de la simulation sur Simulink™. Cette différence de performance est sensible à la façon dont nous avons implanté la dynamique directe. En effet, nous avons utilisé quatre fois la dynamique inverse pour calculer la dynamique directe par la formulation de Newton-Euler tandis que nous avons utilisé seulement deux fois la dynamique inverse pour implanter la dynamique directe sous la formulation de Lagrange-Euler.

Finalement, nous avons obtenu un rendement maximal de 83% pour la simulation en parallèle, ce qui nous permet de simuler le système robotique complet en temps réel avec un pas de calcul de 1ms. Il aurait été possible de séparer le code de simulation du robot sur un plus grand nombre de nœuds de calcul. Par contre, étant donné l'ajout du temps de communication proportionnel an nombre de nœuds et de l'inégalité de la charge de calcul des processeurs qui en résulterait, il est incertain que le pas de calcul aurait pu être diminué. De plus, pour diviser davantage le code de simulation, il aurait fallu approximer certains calculs de nature algébrique car la séparation du modèle introduit des délais de communication. Il est à noter que la séparation actuelle du modèle du robot est valable seulement pour un environnement flexible.

CHAPITRE VI

CONCLUSION ET RECOMMANDATIONS

Nous avons conçu un outil de conception de systèmes robotiques de topologie série permettant d'accélérer grandement le processus de modélisation. Par l'intermédiaire d'un interface graphique, l'usager peut facilement entrer les paramètres numériques nécessaires au générateur de code automatique. Le générateur de code calcule ensuite les équations cinématiques et dynamiques selon la méthode de Newton-Euler. Le résultat de ce calcul peut être écrit sous la formulation de Newton-Euler ou Lagrange. Le générateur de code inclut ensuite les équations dans les fichiers de simulation. L'utilisateur peut ensuite construire son modèle rapidement dans *Simulink*™. La seule étape qui n'est pas automatisée, pour des raisons de non unicité de solution, est la cinématique inverse.

Pour valider l'outil de modélisation, nous l'avons utilisé pour concevoir une main robotique à trois doigts contrôlée en position/force et en interaction avec un objet. Les résultats de simulation montrent que le contrôleur de position identifie bien les masses des doigts de la main. L'erreur de position absolue moyenne reste toujours sous 1mm et l'erreur absolue moyenne d'orientation de l'effecteur demeure sous 0.5° (8.7×10^{-3} rad). L'erreur de force maximale est inférieure à 0.2% de la consigne pour tous les essais, même les essais avec conditions extrêmes. Cette faible erreur de force démontre une robustesse du contrôleur de force à des variations de la rigidité de l'objet d'un facteur d'au moins 10 et à des perturbations de position (lorsque l'objet bouge pendant le contrôle de force). Ce contrôleur de force permet de commander efficacement la force de contact en régime établi. Par contre, nous contrôlons mal le régime transitoire du contact.

Nous avons comparé les algorithmes de Lagrange-Euler et de Newton-Euler du point de vue des avantages et inconvénients rencontrés lors de leur utilisation. Le modèle obtenu avec la formulation de Lagrange-Euler est environ 7 fois plus rapide à simuler sur Simulink™ que par la formulation de Newton-Euler pour le cas étudié. Par contre, le modèle sous la forme de Newton-Euler est plus modulaire et flexible d'utilisation. C'est à dire que nous pouvons plus facilement ajouter de l'information au modèle. Par exemple, nous pouvons directement relier en série deux modèles de robots.

Les contributions apportées par ce projet sont les suivantes. Premièrement, nous avons créé un outil de conception automatisée de systèmes robotiques pour *Matlab®* et *Simulink™*. Cet outil permet de générer automatiquement diverses formes de modèle de robots série. La facilité d'accès aux logiciels (*Windows*, *QNX*, *Matlab®*, *Simulink™*, *Real-Time Workshop™*) et matériaux (PC) utilisés donne un outil flexible et évolutif. Il permet, par exemple, d'afficher le robot en trois dimensions à l'écran et ce, en temps réel ou différé. Pour ce faire, nous pouvons utiliser la routine *Matlab®* qui a été conçue à cet effet ou des logiciels commerciaux de réalité virtuelle. Avec cet outil, nous pouvons aussi utiliser des séparateurs de code, comme celui de Opal-rt Technologies inc. [24] pour simuler le modèle en temps réel sur système de traitement parallèle. Ensuite, nous avons apporté une contribution du côté commande de robots. Nous avons testé la combinaison de contrôleurs position/force en utilisant les algorithmes de Slotine et Li [20] et Seraji [17]. Nous sommes arrivés à commander un manipulateur avec une bonne précision autant en position qu'en force. Nous avons aussi élaboré des stratégies

permettant d'améliorer les performances de systèmes robotiques commandés en position/force et effectuant certaines tâches impliquant un contact avec l'environnement.

Il existe plusieurs possibilités d'amélioration des algorithmes conçus. Nous pourrions améliorer la flexibilité de l'outil pour permettre la modélisation de boucles cinématiques fermées. Nous pourrions aussi concevoir une méthode de séparation de modèle automatique. Du coté de la simulation, nous pourrions modéliser l'environnement comme un ressort non linéaire pour améliorer le réalisme de la simulation. Des stratégies de saisie pourraient être développées pour des objets décentrés ou libres de se déplacer dans toutes les directions. Pour limiter les pics d'erreur de force au début d'un contact, nous pourrions programmer une approche de l'effecteur à vitesse constante jusqu'à ce qu'un contact soit détecté [30]. Finalement, nous pourrions combiner l'algorithme « Compliance control » de Seraji [17] avec notre contrôleur de force de façon à commander le régime transitoire en plus du régime établi du contact.

La démarche ayant mené à l'écriture du générateur de code peut être refaite pour d'autres applications en génie électrique. Par exemple, dans le domaine de la qualité de l'onde, nous voulons étudier l'influence de différents facteurs sur le comportement statistique des harmoniques produits par les charges non linéaires dont le nombre ne cesse de croître dans les réseaux de distribution d'électricité. Nous pourrions produire automatiquement des codes de simulation de différents agencements de convertisseurs, charges et filtres pour finalement simuler le tout en parallèle et en temps réel.

BIBLIOGRAPHIE

[1] S.K.Gupta, C.J.J. Paredis, R. Sinha, C.H. Wang et P.F. Brown, An Intelligent Environment for Simulating Mechanical Assembly Operations, *Proceedings of ASME Design Engineering Technical Conference*, Sept 1998, 1-12.

[2] J. J. Craig, *Introduction to Robotics - Mechanics and control*, Second edition, Addison Wesley Publishing Company, 1989.

[3] J.V.Miro, M. Stoker et R. Gill, The Use of Computer Graphics for Robot Motion Simulation, 11th ISPE/IEE/IFAC International Conference on CAD/CAM, Robotics and Factories of the Future CARS and FOF'95, 2, 1995, pages 884-9.

[4] U. Cugini, M. Ippolito et C. Rizzi, Modeling and Simulation of Handling Machinery with Dynamic and Static Behavior of Non-rigid Materials, *Ieee Robotics and Automation Magazine*, 5(1), 1998, pages 48-56.

[5] L. Sciavicco et B. Siciliano, *Modeling and Control of Robot Manipulators*, (New York; McGraw-Hill Series in Electrical and Computer Engineering, 1996).

[6] A. Abou El-Ela, J. Bohm, R. Isermann, Dynamic model approach for force estimation and control of robotic contact tasks, *MIM-S2 '93. Proceedings of IMACS/IFAC Second Symposium on: Mathematical and Intelligent Models in Systems Simulation*, 11, 1993, pp5-10.

[7] W. Khalil, A system for generating the symbolic models of robots, *Robot Control 1994 (SYROCO'94)*, 2, 1994, pp461-8.

[8] G. Chen, X. Fu, D. Liao, Automatic Generation of Dynamic Model and Simulation for Dual-arm Robot, *Proceedings 2nd Asian Conference on Robotics and its Applications*, 1996, 174-8.

[9] I.M.Chen, G. Yang, Automatic Model Generation for Modular Reconfigurable Robot dynamics, *Transaction of the ASME*, 120, 1998, 346-52.

[10] J.C. Piedboeuf, Recursive Modeling of Serial flexible manipulators, *The Journal of the Astronautical Sciences*, 46(1) : 1-24, 1998.

[11] www.dynasim.se

[12] Dong Sun, Xiaolun Shi, Yunhui Liu, Modelling and cooperation of two-arm robotic system manipulating a deformable object, *Proceedings 1996 IEEE International Conference on Robotics and Automation (Cat. No. 96CH35857)*, 3, 1996, pages 2346-51.

[13] D.E. Whitney, Force Feedback Control of Manipulator Fine Motions, *ASMEJ Dyn.Sys.Meas.Control*, 1977, pp91-97.

[14] M. Raibert, J. Craig, Hybrid Position/Force Control of Manipulators, *ASME J. Dyn. Sys. Meas. Control*, 102(2), 1981, pp126-33.

[15] N. Hogan, Impedance Control : An Approach to Manipulation, parts I-III, *ASME J. Dyn. Sys. Meas. Control*, 107(1), 1985, pp1-24.

[16] S.T. Lin, K.H.Yae, Identification of Unknown Payload and Environmental Parameters for Robot Compliant Motion, *Proceedings of the 1992 American Control Conference (IEEE Cat. No.92CH3072-6)*, 4, 1992, pp2952-6.

[17] H. Seraji, Nonlinear and Adaptive-Control of Force and Compliance in Manipulators, *International Journal of Robotics Research*, 17(5), 1998, pp467-84.

[18] M. Vukobravic, R. Stojik, Y. Ekalo, Contribution to the Position/Force Control of Manipulation Robots interacting with Dynamic Environment - A Generalisation, *Automatica,* 34(10), 1998, pp1219-26.

[19] S.R.Pandian, S. Kawamura, Hybrid Force/Position Control for Robot Maniulators based on a D-type Learning Law, *Robotica,* 14(1), 1996, pp51-9.

[20] J.-J.E. Slotine et W. Li, *Applied Nonlinear Control*, (Englewood cliffs NJ; Prentice Hall, 1991).

[21] J.J. Zhang, Y.F. Lu, B. Wang, A Nonrecursive Newton-Euler Formulation for the Parallel Computation of Manipulator Inverse Dynamics, *IEEE Transactions on Systems Man and Cybernetics Part C - Applications and Reviews*, 28(3), 1998, pp467-71.

[22] E. Abdalla, H.J. Pu, M. Muller, A.A. Tantawy, L. Abdelatif, H. Nour Eldin, A Novel Parallel Recursive Newton-Euler Algorithm for Modelling and Computation of Robot Dynamics, *Mathematics and Computers in Simulation*, 37(2-3), 1994, pp227-40.

[23] H.J. Pu, M. Muller, E. Abdalla, L. Abdelatif, E. Bakr, H.A. Nour Eldin, Parallel Computation of the Inertia Matrix of a Tree Type Robot Using One Directional Recursion of Newton-Euler Formulation, *Journal of Intelligent and Robotic Systems : Theory and Applications*, 15(1), 1996, pp33-9.

[24] Opal-RT Technologies, *Description du séparateur de code*, (Montréal ; Opal-rt Technologies Inc., Document interne, 1999).

[25] G. Ferretti, Systematic dynamic modelling of mechanical systems containing kinematic loops, *Mathematical Modelling of Systems*, 2(3), 1996, pp212-35.

[26] S. Megahed et M. Redaud, *Dynamic modeling of Robot Manipoulators Containing Closed Kinematic Chains, in Advanced Software in Robotics*, A. Danthine et M. Géradin (éds), © Elsevier Science Publishers B.V., Hollande, 1984, pp147-58.

[27] M. de Montigny, P. Sicard, Résolution symbolique de la cinématique directe d'un manipulateur avec N joints à l'aide de Matlab®-Maple™, Université du Québec à Trois-Rivières, Groupe de recherche en électronique industrielle, Rapport de recherche interne, 1994.

[28] M. de Montigny, P. Sicard, Résolution symbolique de la dynamique inverse et directe d'un manipulateur avec N joints à l'aide de Matlab®-Maple™, Université du Québec à Trois-Rivières, Groupe de recherche en électronique industrielle, Rapport de recherche interne, 1995.

[29] Neil F. Stewart et Francis Jensen, Solutions numériques des problèmes matriciels, Les presses de l'Université de Montréal, Édition Eyrolles-Paris, 1975.

[30] Dimitri M. Gorinevsky, Alexander M. Formalsky et Anatoly Yu. Schneider, Force Control of Robotics Systems, Library of Congress Cataloging-in-Publication Data, CRC Press LLC, 1997.

[31] M. de Montigny, P. Sicard, Commande adaptative force/position d'une main robotique, *Proceedings of 1999 IEEE Canadian Conference on Electrical and Computer Engineering*, Edmonton, Canada, May 1999.

[32] M. de Montigny, P. Sicard, Automatic dynamic model generation for rigid robot simulation and control, *Proceedings of the IASTED International Conference, Modelling and Simulation*, Philadelphia, USA, May 1999.

Annexe A

Programmes de génération de code

Programme principal de génération de code sous la formulation de Lagrange-Euler

```
function
[Gravdir,ki,alphaim1,aim1,di,thetai,masse,ccmdh,typten,param,tenseur,n,
...

nomnou,N,M,fv,rbase,rotbase,posbase,propcin,paramcin,Prop_ne_vit,Prop_ne
_mas,ctes,pcidi]= ...

genalgo_le(choix,Gravdir,ki,alphaim1,aim1,di,thetai,masse,ccmdh,typten,p
aram, ...

tenseur,n,nomnou,N,M,fv,rbase,rotbase,posbase,propcin,paramcin,Prop_ne_v
it,Prop_ne_mas,ctes,pcidi);
% genalgo_le : Génération de code automatique d'un robot et de son
contrôleur de position
%
% Ce programme est appelé par "finitr"  (programme de gestion de
l'interface graphique du
% générateur de code).  Ce programme utilise les programmes cindir et
dynam de calcul symbolique
% de la cinématique directe et de la dynamique du robot.
%
% Par : Martin de Montigny
% Dernière modification : 14 octobre 1999 (debug matc et matmp)

   % précision
   digits(12);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Assignation de variables automatiquement en fonction des paramètres
   % cinématiques et dynamiques numériques

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Création du début des variables
   sdi='[';
   sthetai='[';
   salphaim1='[';
   saim1='[';
   sccmdh='';
   slongueur='';
   stenseur='';
   smasse='';
   smasse2='';
   smasse_tenseur='';
   vardefbra='syms ';
   vardefman1='syms ';
   vardefman2='syms DPU ';

   % Recherche des longueurs des membres
   compteurL=0;
```

```
   !del fichiernum.mat; % Fichier qui contiendra les valeurs numériques
des constantes créées
   for a=1:N
      flag=0;
      if ((ki(a)==0)&(a<=M))
         eval(['syms d',num2str(a),' real']);
         vardefbra=[vardefbra,'d',num2str(a),' '];
         eval(['sdi=[sdi,''d',num2str(a),''']];']);
         flag=1;
      end
      if ((ki(a)==0)&(a<=M)&(di(a)~=0))
         sdi=[sdi,'+'];
      end
      if (di(a)~=0)
         % Ajout de la constante numérique à fichiernum.mat
         eval(['L',num2str(compteurL),'=',num2str(di(a)),';']); % Valeur
numérique de la constante créée
         if (length(dir('fichiernum.mat')))
            eval(['save fichiernum L',num2str(compteurL),' -append']);
         else
            eval(['save fichiernum L',num2str(compteurL)]);
         end

         eval(['syms L',num2str(compteurL),' real']);
         vardefbra=[vardefbra,'L',num2str(compteurL),' '];
         % La variable symbolique sdi est identique (en structure) à la
variable
         % di sauf que les valeurs sont remplacées par des variables
symboliques,
         % le tout sous forme de chaine de caractère (sdi)...
         eval(['sdi=[sdi,''L',num2str(compteurL),''']];']);
         % Pour pouvoir se souvenir de la valeur numérique des variables
symboliques,
         % nous devons placer ces valeurs numériques dans des vecteurs
(la valeur ayant
         % la position 1 dans ce vecteur corresond à L0...)
         ndi(compteurL+1,1)=di(a);
         compteurL=compteurL+1;
         flag=1;
      end
      if (flag==0)
         sdi=[sdi,'0;'];
      else
         sdi=[sdi,';'];
      end


      if (aim1(a)~=0)
         eval(['L',num2str(compteurL),'=',num2str(aim1(a)),';']); %
Valeur numérique de la constante créée
         % Ajout de la constante numérique à fichiernum.mat
         if (length(dir('fichiernum.mat')))
            eval(['save fichiernum L',num2str(compteurL),' -append']);
         else
            eval(['save fichiernum L',num2str(compteurL)]);
         end
```

```
            eval(['syms L',num2str(compteurL),' real']);
            vardefbra=[vardefbra,'L',num2str(compteurL),' '];
            eval(['saim1=[saim1,''L',num2str(compteurL),';''];']);
            naim1(compteurL+1,1)=aim1(a);
            compteurL=compteurL+1;
         else
            saim1=[saim1,'0;'];
         end
      end
      % Enlève le dernier ";" qui ne sert à rien et ajoute le "];" final...
      sdi=[sdi(1:length(sdi)-1),'];'];
      saim1=[saim1(1:length(saim1)-1),'];'];

      for a=1:N
         flag=0;
         if ((ki(a)==1)&(a<=M))
            eval(['syms T',num2str(a),' real']);
            vardefbra=[vardefbra,'T',num2str(a),' '];
            eval(['sthetai=[sthetai,''T',num2str(a),''''];']);
            flag=1;
         end
         if ((flag==1)&(thetai(a,1)~=0))
            sthetai=[sthetai,'+'];
         end
         if (thetai(a,1)~=0)
            syms temp real
            temp=vpa(thetai(a,1)); %pour garder la structure pi/x si
possible...
            if (strcmp(class(temp),'sym'))
               sthetai=[sthetai,strvcat(vpa(temp))];
            else
               sthetai=[sthetai,strvcat(vpa(temp))]; %num2str avant...
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
            end
            flag=1;
         end
         if (flag==0)
            sthetai=[sthetai,'0;']; %s'il n'y a aucune variable dans
thetai, alors écrire 0
         else
            sthetai=[sthetai,';']; %sinon mettre le ";" de séparation quand
même...
         end

         if (alphaim1(a,1)~=0)
            temp=sym(alphaim1(a,1)); %pour garder la structure pi/x si
possible...
            if (strcmp(class(temp),'sym'))
               salphaim1=[salphaim1,strvcat(vpa(temp)),';'];
            else
               salphaim1=[salphaim1,strvcat(vpa(temp)),';']; % num2str
avant...!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
            end
         else
            salphaim1=[salphaim1,'0;'];
         end
      end
```

```
   sthetai=[sthetai(1:length(sthetai)-1),'];'];
   salphaim1=[salphaim1(1:length(salphaim1)-1),'];'];
   vardefbra=[vardefbra,'real'];

   % Recherche des tenseurs (s'il y en a)
   for a=0:(N-1)
      stenseurt='';
      for b=1:3
         for c=1:3
            if (tenseur(a*9+(b-1)*3+c,1)~=0)
               % Valeur numérique de la constante créée

eval(['t',num2str(a+1),num2str(b),num2str(c),'=',num2str(tenseur(a*9+(b-
1)*3+c,1)),';']);
               % Ajout de la constante numérique à fichiernum.mat
               if (length(dir('fichiernum.mat')))
                  eval(['save fichiernum
t',num2str(a+1),num2str(b),num2str(c),' -append']);
               else
                  eval(['save fichiernum
t',num2str(a+1),num2str(b),num2str(c)]);
               end

               % Norme des variables symboliques représentant les
éléments des tenseurs :
               % txyz : x=numéro du tenseur (membre)
               %        y=ligne dans le tenseur
               %        z=colonne dans le tenseur

               eval(['syms t',num2str(a+1),num2str(b),num2str(c),'
real']);

eval(['stenseurt=[stenseurt,''t',num2str(a+1),num2str(b),num2str(c),',''
];']);
               % Pas besoin de ntenseur car les variables txyz
représentent de façon
               % implicite la position des valeurs numériques
représentées par les
               % variables txyz.
            else
               stenseurt=[stenseurt,'0,'];
            end
         end
      end
      stenseur=strvcat(stenseur,stenseurt);
   end

   % Recherche des masses
   compteurm=0;
   for a=1:M
      if (masse(a)~=0)
         eval(['M',num2str(compteurm),'=',num2str(masse(a)),';']); %
Valeur numérique de la constante créée
         % Ajout de la constante numérique à fichiernum.mat
         if (length(dir('fichiernum.mat')))
            eval(['save fichiernum M',num2str(compteurm),' -append']);
         else
```

```
            eval(['save fichiernum M',num2str(compteurm)]);
         end
         eval(['syms M',num2str(compteurm),' real']);
         eval(['smasse=[smasse,''M',num2str(compteurm),' ''];']);

eval(['smasse2=strvcat(smasse2,''M',num2str(compteurm),''');']);
         nmasse(compteurm+1,1)=masse(a);

eval(['smasse_tenseur=strvcat(smasse_tenseur,''M',num2str(compteurm),',
');']);
         compteurm=compteurm+1;
      else
         smasse_tenseur=strvcat(smasse_tenseur,'0,');
      end
   end


   % Recherche des positions des masses (paramètres dh mod)
   compteurl=0;
   sccmdh='';

   for a=1:M
      sccmdht='';
      syms temp real
      temp(1,1)=ccmdh(a,1);
      sccmdht=[strvcat(temp),','];
      if (ccmdh(a,2)~=0)
         eval(['l',num2str(compteurl),'=',num2str(ccmdh(a,2)),';']); %
Valeur numérique de la constante créée
         % Ajout de la constante numérique à fichiernum.mat
         if (length(dir('fichiernum.mat')))
            eval(['save fichiernum l',num2str(compteurl),' -append']);
         else
            eval(['save fichiernum l',num2str(compteurl)]);
         end

         eval(['syms l',num2str(compteurl),' real']);
         eval(['sccmdht=[sccmdht,''l',num2str(compteurl),','''];']);
         eval(['slongueur=[slongueur,''l',num2str(compteurl),' ''];']);
         nccmdh(compteurl+1,1)=ccmdh(a,2);
         compteurl=compteurl+1;
      else
         sccmdht=[sccmdht,'0,'];
      end
      if
(ccmdh(a,3)~=0)%*********************************************************
*******mettre des ;...
         eval(['l',num2str(compteurl),'=',num2str(ccmdh(a,3)),';']); %
Valeur numérique de la constante créée
         % Ajout de la constante numérique à fichiernum.mat
         if (length(dir('fichiernum.mat')))
            eval(['save fichiernum l',num2str(compteurl),' -append']);
         else
            eval(['save fichiernum l',num2str(compteurl)]);
         end

         eval(['syms l',num2str(compteurl),' real']);
```

```
            eval(['sccmdht=[sccmdht,''l'',num2str(compteurl),','']']);
            eval(['slongueur=[slongueur,''l'',num2str(compteurl),' ''];']);
            nccmdh(compteurl+1,1)=ccmdh(a,3);
            compteurl=compteurl+1;
         else
            sccmdht=[sccmdht,'0,'];
         end
         temp(1,1)=ccmdh(a,4);
         sccmdht=[sccmdht,[strvcat(temp),',']];

         sccmdh=strvcat(sccmdh,sccmdht);
      end

      % Création de ki en format de chaime
      ski='[';
      for a=1:M
         if ki(a)==0
            ski=[ski,'0;'];
         else
            ski=[ski,'1;'];
         end
      end
      ski=[ski(1:length(ski)-1),'];'];



%%%
```

```
   tabdefbra=['N=',num2str(N)]; % note : ????il n'y a pas d'erreur : M
de defbra = N de initr (conflit de notation)
   chaine=['M=',num2str(M)];
   tabdefbra=strvcat(tabdefbra,chaine);
   chaine=['Alphaim1=',strvcat(salphaim1)];
   tabdefbra=strvcat(tabdefbra,chaine);
   chaine=['Aim1=',strvcat(saim1)];
   tabdefbra=strvcat(tabdefbra,chaine);
   chaine=['Di=',strvcat(sdi)];
   tabdefbra=strvcat(tabdefbra,chaine);
   chaine=['Thetai=',strvcat(sthetai)];
   tabdefbra=strvcat(tabdefbra,chaine);
   chaine=['Ki=',ski];
   tabdefbra=strvcat(tabdefbra,chaine);

   h=fopen('temp.txt','W');
   fprintf(h,'%s\n\n',vardefbra);
   for a=1:7
      fprintf(h,'%s\n',tabdefbra(a,:));
   end
   fclose(h);

   eval(['!perl moddefbra.pl ',nomdefbra,' ',nomcin,' ',nomsave]); %
Création de defbrax.m


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
   % Création des fichiers temporaires temp1.txt et temp2.txt pour la
création de defman %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%

   % Écriture de ccmdh dans fichier temp1.txt
   eval(['tabdefman1=[''syms '',slongueur,''real'']']);
   tabdefman1=strvcat(tabdefman1,['ccmdh = [',sccmdh(1,:),'...']);
   for a=2:(M-1)
      tabdefman1=strvcat(tabdefman1,['         ',sccmdh(a,:),'...']);
   end
   tabdefman1=strvcat(tabdefman1,['
',sccmdh(M,1:(length(sccmdh(M,:))-1)),'];']);
   h=fopen('temp1.txt','W');
   for a=1:(M+1)
      fprintf(h,'%s\n',tabdefman1(a,:));
   end
   fclose(h);


   % Écriture des tenseurs dans temp2.txt
   tabdefman2=['syms DPU ',smasse,'real']
   tabdefman2=strvcat(tabdefman2,['Masi =
[DPU,',smasse_tenseur(1,:),stenseur(1,:),'...']);
   for a=2:(M-1)
      tabdefman2=strvcat(tabdefman2,['
DPU,',smasse_tenseur(a,:),stenseur(a,:),'...']);
   end
```

```
   tabdefman2=strvcat(tabdefman2,['
DPU,',smasse_tenseur(M,:),stenseur(M,1:length(stenseur(M,:))-1),'];']);
   h=fopen('temp2.txt','W');
   for a=1:(M+1)
      fprintf(h,'%s\n',tabdefman2(a,:));
   end
   fclose(h);

   % Écriture du code de définition du vecteur opposé à la gravité (pour
inclure l'effet de la gravité)
   h=fopen('temp3.txt','W');
   tabdefman3=['vp(1,1)=vpa(',num2str(-Gravdir(1)),');'];
   tabdefman3=strvcat(tabdefman3,['vp(2,1)=vpa(',num2str(-
Gravdir(2)),');']);
   tabdefman3=strvcat(tabdefman3,['vp(3,1)=vpa(',num2str(-
Gravdir(3)),');']);
   for a=1:3
      fprintf(h,'%s\n',tabdefman3(a,:));
   end
   fclose(h);


   eval(['!perl moddefman.pl ',nomdefman,' ',nomsave,' ',num2str(M)]); %
Création de defmanx.m

   % Création du fichier dynx d'exécution de dynam
   eval(['h=fopen(''',nomdyn,'.m'',''W'')']);
   fprintf(h,'%s\n','% DYNx est le fichier exécuteur de la dynamique
inverse');
   fprintf(h,'%s\n','% x représente le type de manipulateur dont on a
auparavant calculé');
   fprintf(h,'%s\n','% la cinématique directe avec CINx');
   fprintf(h,'%s\n','% ');


eval(['fprintf(h,''%s\n\n'',''fichier=([''''',nomdefman,''''']);'');']);
   fprintf(h,'%s\n\n','[Ti,MTheta,gTheta,VThThp]=dynam(fichier);');
   eval(['fprintf(h,''%s\n'',''save ',nomsavedyn,''');']);
   fclose(h);


   %%%%
```

```
   % Création de matc (matrice de Cristoffel) et de matmp (dérivée de la
matrice des masses)
   for a=1:M
      if (ki(a)==1)
         eval(['syms Tp',num2str(a),' real']);
      else
         eval(['syms Dp',num2str(a),' real']);
      end
   end

   % Calcul de la matrice de Cristoffel (termes centrifuges et de
Coriolis)
   matc=sym(zeros(M));
   for kk=1:M
      for jj=1:M
         for ii=1:M
            temp=['matc(kk,jj)=matc(kk,jj)+0.5*(diff(MTheta(kk,jj),''''];
            if (ki(ii)==1)
               temp=[temp,'T',num2str(ii),''')+'];
            else
               temp=[temp,'d',num2str(ii),''')+'];
            end
            temp=[temp,'diff(MTheta(kk,ii),''''];
            if (ki(jj)==1)
               temp=[temp,'T',num2str(jj),''')-'];
            else
               temp=[temp,'d',num2str(jj),''')-'];
            end
            temp=[temp,'diff(MTheta(ii,jj),''''];
            if (ki(kk)==1)
               temp=[temp,'T',num2str(kk),''''))*'];
            else
               temp=[temp,'d',num2str(kk),''''))*'];
            end
            if (ki(ii)==1)
               temp=[temp,'Tp',num2str(ii),';'];
            else
               temp=[temp,'Dp',num2str(ii),';'];
            end
            eval(temp);
         end
      end
   end

   % Calcul de la dérivée de la matrice des masses/inerties par rapport
au temps
   % Construction de la chaine de car. de dérivée en chaine
   temp='matmp=';
   for a=1:M
      if (ki(a)==1)

temp=[temp,'diff(MTheta,''T',num2str(a),''')*Tp',num2str(a),'+'];
      else

temp=[temp,'diff(MTheta,''d',num2str(a),''')*Dp',num2str(a),'+'];
      end
   end
```

```
   temp=temp(1:length(temp)-1); %enlève le + de trop
   temp=[temp,';'];
   eval(temp);
   matmp=simple(matmp);

   %%%%
```

```
   % Remplace les constantes multipliées entre elles par des constantes
précalculées
   slistec='';
   nlistec=[];


[MTheta,slistec,nlistec]=remplace_constante(MTheta,slistec,nlistec,'fich
iernum');

[VThThp,slistec,nlistec]=remplace_constante(VThThp,slistec,nlistec,'fich
iernum');

[gTheta,slistec,nlistec]=remplace_constante(gTheta,slistec,nlistec,'fich
iernum');

[matc,slistec,nlistec]=remplace_constante(matc,slistec,nlistec,'fichiern
um');

[matmp,slistec,nlistec]=remplace_constante(matmp,slistec,nlistec,'fichie
rnum');

   % Seulement pour trouver les constantes utilisées...

[JVid0t,slistec,nlistec]=remplace_constante(JVid0t,slistec,nlistec,'fich
iernum');

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Génération de code cinématique et dynamique %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Transformation des matrices symboliques en chaines de caractères

   % Définition (en symbolic) des variables comprises dans MTheta,
VThThp et gTheta
   for a=1:M
      for b=1:M
         temp2=findsym(MTheta(a,b));
         temp=strrep(temp2,',','');
         eval(['syms ',temp,' real']);

eval(['sMTheta',num2str(a),num2str(b),'=''',strvcat(MTheta(a,b)),''';'])
;
      end
   end
   for a=1:M
      temp2=findsym(VThThp(a));
      temp=strrep(temp2,',','');
      eval(['syms ',temp,' real']);
      eval(['sVThThp',num2str(a),'=''',strvcat(VThThp(a)),''';']);
   end
   for a=1:M
      temp2=findsym(gTheta(a));
      temp=strrep(temp2,',','');
      eval(['syms ',temp,' real']);
      eval(['sgTheta',num2str(a),'=''',strvcat(gTheta(a)),''';']);
```

```
   end

   % Remplacement des nombres par des "nombres.0"
   for a=1:M
      for b=1:M
         % Met terme dans une variable temporaire pour faciliter le
traitement

eval(['sMThetat=ccode(MTheta(',num2str(a),',',num2str(b),'));']);
         sMThetat=sMThetat(12:length(sMThetat)-1);
         sMThetat=strrep(sMThetat,'~','');
         eval(['sMTheta',num2str(a),num2str(b),'=sMThetat']);
      end
   end

   for a=1:M
      % Met terme dans une variable temporaire pour faciliter le
traitement
      eval(['sVThThpt=ccode(VThThp(',num2str(a),'));']);
      sVThThpt=sVThThpt(12:length(sVThThpt)-1);
      sVThThpt=strrep(sVThThpt,'~','');
      eval(['sVThThp',num2str(a),'=sVThThpt']);
   end

   for a=1:M
      % Met terme dans une variable temporaire pour faciliter le
traitement
      eval(['sgThetat=ccode(gTheta(',num2str(a),'));']);
      sgThetat=sgThetat(12:length(sgThetat)-1);
      sgThetat=strrep(sgThetat,'~','');
      eval(['sgTheta',num2str(a),'=sgThetat']);
   end

   % Matrice des masses
   % Remplacement des variables des équations par des u[x]

   % Initialisation du compteur d'entrée
   um=M; %rendu après les variables articulaires de position (après di)

   % Assignation des entrées à position constante

   % di (0 à M-1)
   % Les "M" premières entrées sont les variables articulaires
(position) peu importe si
   % ce sont les di (distances) ou Thetai (angles).
   for c=1:M
      for a=1:M
         for b=1:M
            % Production de la chaine représentant l'entrée pour une
var. art.
            eval(['umt=''u[',num2str(c-1),']'';']);
            % Remplacement des "dx" par u[um]

eval(['sMTheta',num2str(a),num2str(b),'=strrep(sMTheta',num2str(a),num2s
tr(b), ...
                  ',''d',num2str(c),''',''',umt,''');']);
         end
```

```
      end
   end
   % masses
   % Les masses sont répertoriés dans le tableau smasse2 ; chaque ligne
est une masse
   longm=size(smasse2); % nombre de lignes du répertoire des masses du
robot
   for c=1:longm(1)
      for a=1:M
         for b=1:M
            eval(['umt=''u[',num2str(um),']'';']); % Production de la
chaine représentant l'entrée pour une masse.

eval(['sMTheta',num2str(a),num2str(b),'=strrep(sMTheta',num2str(a),num2s
tr(b), ...
                  ',''',deblank(smasse2(c,:)),''',''',umt,''');']);
         end
      end
      % Incrément du compteur d'entrée
      um=um+1;
   end


   % vecteur sin cos (de 2M à ?)
   % slistet : liste des fonctions trigo en entrée (1 fonction par
ligne)
   longt=size(slistet);
   for c=1:longt(1)
      eval(['umt=''u[',num2str(um+c-1),']'';']); % Production de la
chaine représentant l'entrée pour un terme trigo
      for a=1:M
         for b=1:M

eval(['sMTheta',num2str(a),num2str(b),'=strrep(sMTheta',num2str(a),num2s
tr(b), ...
                  ',''',deblank(slistet(c,:)),''',''',umt,''');']);
         end
      end
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Écriture du code de calcul de la matrice des masses %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   h=fopen('dyndirle_m.c','w') % fichier destination
   h2=fopen('dyndirle_m_s.c','r') % fichier source

   % Écriture du début de dyndir_le_m
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*1*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Définition des constantes
   longc=size(slistec);
```

```
   for a=1:longc(1)
      if (mod(nlistec(a),1)~=0) % pour savoir si on ajoute le .0 après
la valeur numérique
         fprintf(h,'%s\n',['#define ',deblank(slistec(a,:)),'
',num2str(nlistec(a))]);
      else
         fprintf(h,'%s\n',['#define ',deblank(slistec(a,:)),'
',num2str(nlistec(a)),'.0']);
      end
   end
   fprintf(h,'%s\n',''); % espace

   % Écriture partielle de dyndir_le_m
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*2*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Nombre d'entrées et de sorties
   nent=2*M+longt(1); % calcul du nombre d'entrées
   nsor=M*M; % calcul du nombre de sorties
   fprintf(h,'%s\n',['    ssSetNumInputs(S,',num2str(nent),');']);
   fprintf(h,'%s',['    ssSetNumOutputs(S,',num2str(nsor),');']);

   % Écriture partielle de dyndir_le_m
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*3*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Algorithme
   % p.s. la matrice des masses est symétrique...
   for a=1:M
      for b=1:a
         eval(['sMThetat=sMTheta',num2str(a),num2str(b)]);
         eval(['temp=''y[',num2str((a-1)*M+b-1),']=',sMThetat,';'';']);
         fprintf(h,'%s\n',temp);
      end
   end
   for a=1:(M-1)
      for b=(a+1):M
         eval(['temp=''y[',num2str((a-1)*M+b-1),']=','y[',num2str((b-
1)*M+a-1),'];'';']);
         fprintf(h,'%s\n',temp);
      end
   end

   % Écriture de la fin de dyndir_le_m
   temp=fscanf(h2,'%c',1);
   while (~strcmp(temp,''))
      fprintf(h,'%c',temp);
      temp=fscanf(h2,'%c',1);
   end
```

```
   fclose(h2);
   fclose(h);


   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Bloc de calcul des termes de vitesse/coriolis et de gravité %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Remplacement des variables des équations par des u[x]

   % Initialisation du compteur d'entrée
   um=2*M; %rendu après les variables articulaires de position et de
vitesse

   % Assignation des entrées à position constante

   % di (0 à M-1)
   % Les "M" premières entrées sont les variables articulaires
(position) peu importe si
   % ce sont les di (distances) ou Thetai (angles).
   for c=1:M %c=# de d
      for a=1:M %a=# d'élément de VThThp
         % Production de la chaine représentant l'entrée pour une var.
art.
         eval(['umt=''u[',num2str(c-1),']'';']);
         % Remplacement des "dx" par u[um]
         eval(['sVThThp',num2str(a),'=strrep(sVThThp',num2str(a), ...
               ','''d',num2str(c),''','''',umt,''');']);
         eval(['sgTheta',num2str(a),'=strrep(sgTheta',num2str(a), ...
               ','''d',num2str(c),''','''',umt,''');']);
      end
   end
   % Tpi (M à 2M-1)
   % Les entrées de M à 2M-1 sont les variables articulaires de vitesse
   for c=1:M
      for a=1:M
         % Production de la chaine représentant l'entrée pour une var.
art.
         eval(['umt=''u[',num2str(c-1+M),']'';']);
         % Remplacement des "Tpx" par u[um]
************************************************************
         eval(['sVThThp',num2str(a),'=strrep(sVThThp',num2str(a), ...
               ','''Tp',num2str(c),''','''',umt,''');']);
         eval(['sgTheta',num2str(a),'=strrep(sgTheta',num2str(a), ...
               ','''Tp',num2str(c),''','''',umt,''');']);
         eval(['sVThThp',num2str(a),'=strrep(sVThThp',num2str(a), ...
               ','''Dp',num2str(c),''','''',umt,''');']);
         eval(['sgTheta',num2str(a),'=strrep(sgTheta',num2str(a), ...
               ','''Dp',num2str(c),''','''',umt,''');']);
      end
   end

   % masses
   % Les masses sont répertoriés dans le tableau smasse2 ; chaque ligne
est une masse
```

```
   longm=size(smasse2); % nombre de lignes du répertoire des masses du
robot
   for c=1:longm(1)
      for a=1:M
         eval(['umt=''u[',num2str(um),']'';']); % Production de la
chaine représentant l'entrée pour une masse.
         eval(['sVThThp',num2str(a),'=strrep(sVThThp',num2str(a), ...
               ',''',deblank(smasse2(c,:)),''',''',umt,''');']);
         eval(['sgTheta',num2str(a),'=strrep(sgTheta',num2str(a), ...
               ',''',deblank(smasse2(c,:)),''',''',umt,''');']);
      end
      % Incrément du compteur d'entrée
      um=um+1;
   end


   % vecteur sin cos (de 3M à ?)
   % slistet : liste des fonctions trigo en entrée (1 fonction par
ligne)
   longt=size(slistet);
   for c=1:longt(1)
      eval(['umt=''u[',num2str(um+c-1),']'';']); % Production de la
chaine représentant l'entrée pour un terme trigo
      for a=1:M
         eval(['sVThThp',num2str(a),'=strrep(sVThThp',num2str(a), ...
               ',''',deblank(slistet(c,:)),''',''',umt,''');']);
         eval(['sgTheta',num2str(a),'=strrep(sgTheta',num2str(a), ...
               ',''',deblank(slistet(c,:)),''',''',umt,''');']);
      end
   end



%%%
```

```
         fprintf(h,'%s\n',['#define ',deblank(slistec(a,:)),'
',num2str(nlistec(a))]);
      else
         fprintf(h,'%s\n',['#define ',deblank(slistec(a,:)),'
',num2str(nlistec(a)),'.0']);
      end
   end
   fprintf(h,'%s\n',''); % espace

   % Écriture partielle de dyndir_le_vg
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*2*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Nombre d'entrées et de sorties
   nent=3*M+longt(1); % calcul du nombre d'entrées
   nsor=M; % calcul du nombre de sorties
   fprintf(h,'%s\n',['    ssSetNumInputs(S,',num2str(nent),');']);
   fprintf(h,'%s',['    ssSetNumOutputs(S,',num2str(nsor),');']);

   % Écriture partielle de dyndir_le_vg
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*3*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Algorithme
   % p.s. la matrice des masses est symétrique...
   for a=1:M

eval(['sVg=[sVThThp',num2str(a),','''+('',sgTheta',num2str(a),','');''];'
]);
      eval(['temp=''y[',num2str(a-1),']=',sVg,''';']);
      fprintf(h,'%s\n',temp);
   end

   % Écriture de la fin de dyndir_le_vg
   temp=fscanf(h2,'%c',1);
   while (~strcmp(temp,''))
      fprintf(h,'%c',temp);
      temp=fscanf(h2,'%c',1);
   end

   fclose(h2);
   fclose(h);

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %Compilation des codes générés%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   mex dyndirle_m.c
   mex dyndirle_vg.c
```

```
%%%
```

```
   eval(temp);

% Calcul partiel de la dérivée de P

   dp1=p*w.'*w*p;
   dp1=simple(dp1);
   dp1=simple(dp1);
   dp1=simple(dp1);

   % Définition de modp (module de P)
   temp='ModP=';
   for a=0:M+1:M*M-1
      temp=[temp,'p',num2str(a),'+'];
   end
   temp=[temp(1:length(temp)-1),';']; %enlève le dernier +
   eval(temp);

   dp=p*rho*(1-ModP/k0)-dp1;
   dp=simple(dp);

   % Création du fichier d'équations
   h=fopen('temp.txt','w');
   for a=1:M
      for b=1:M
         temp=strvcat(dp(a,b)); % Extraction de l'élément à écrire
         % Remplacement des éléments P par les entrées
         for c=0:M*M-1

eval(['temp=strrep(temp,''p',num2str(c),''','''u[',num2str(c+M*M),']'');'
]);
         end
         % Remplacement des éléments W par les entrées
         for c=0:M*M-1

eval(['temp=strrep(temp,''w',num2str(c),''','''u[',num2str(c),']'');']);
         end
         % Remplacement des autres variables par les entrées
         temp=strrep(temp,'rho',['u[',num2str(2*M*M),']']); %rho
         temp=strrep(temp,'k0',['u[',num2str(2*M*M+1),']']); %k0

         % Ajout du 'y[x]='
         temp=['y[',num2str((a-1)*M+b-1),']=',temp,';'];

         fprintf(h,'%s\n',temp);
      end
   end
   fclose(h);

   % Construction du fichier C de calcul de P'
*****************************
   clear h
   h1=fopen('temp.txt','r'); % fichier d'équations
   h2=fopen('calcul_pp_s.c','r'); % fichier source sans équations
   h3=fopen('calcul_pp.c','w'); % fichier destination

   % Écriture du début du code
   temp2=fscanf(h2,'%c',5);
```

```
   while (~strcmp(temp2,'/*1*/'))
      fprintf(h3,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Écriture du nombre d'entrées et de sorties
   fprintf(h3,'%s\n',['    ssSetNumInputs(        S,
',num2str(2*M*M+2),'); /* number of inputs */']);
   fprintf(h3,'%s\n',['    ssSetNumOutputs(       S, ',num2str(M*M),');
/* number of outputs */']);

   % Écriture partielle du code
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*2*/'))
      fprintf(h3,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Écriture des équations
   temp=fscanf(h1,'%c',1);
   while (~strcmp(temp,''))
      fprintf(h3,'%c',temp);
      if (temp~=';')
         temp=fscanf(h1,'%c',1);
      else
         temp=fscanf(h1,'%c\n',1);
      end
   end

   % Écriture de la fin du code
   temp=fscanf(h2,'%c',1);
   while (~strcmp(temp,''))
      fprintf(h3,'%c',temp);
      temp=fscanf(h2,'%c',1);
   end

   % Fermeture des fichiers
   fclose(h3);
   fclose(h2);
   fclose(h1);

   % Calcul de W
**********************************************************************
*******

   % Note : nous avons créé précédemment les matrices matc (matrice de
Cristoffel) et matmp
   %        (dérivée de la matrice des masses)

   %définition dynamique des vecteurs de vitesse articulaire et de masse
   temp='[';
   temp2='[';
   for a=1:M
      if (ki(a)==1)
         temp=[temp,'Tp',num2str(a),';'];
```

```
         eval(['syms Tp',num2str(a),' real']);
      else
         temp=[temp,'Dp',num2str(a),';'];
         eval(['syms Dp',num2str(a),' real']);
      end
      temp2=[temp2,'M',num2str(a-1),';'];
      eval(['syms M',num2str(a-1),' real']);
   end
   temp=[temp(1:length(temp)-1),']'];
   eval(['vectqp=',temp,';']);
   temp2=[temp2(1:length(temp2)-1),']'];
   eval(['vectM=',temp2,';']);

   syms lamda real

   w00=(matc-matmp)*vectqp+gTheta; % à filtrer
**************************************************
   w01=-MTheta*vectqp; % à dériver et filtrer
   w02=MTheta*vectqp*lamda; % tel quel
   w00=simple(w00);
   w01=simple(w01);
   w02=simple(w02);

   w00=jacobian(w00,vectM);
   w01=jacobian(w01,vectM);
   w02=jacobian(w02,vectM);

   w00=simple(w00);
   w01=simple(w01);
   w02=simple(w02);

   % Remplacement des variables par des entrées
   for a=1:M
      for b=1:M
         % Transformation des variables symboliques en chaines de
caractères
         eval(['sw00=w00(a,b);']);
         sw00=ccode(sw00);
         sw00=sw00(12:length(sw00)-1);
         sw00=strrep(sw00,'~','');
         eval(['sw01=w01(a,b);']);
         sw01=ccode(sw01);
         sw01=sw01(12:length(sw01)-1)
         sw01=strrep(sw01,'~','');
         eval(['sw02=w02(a,b);']);
         sw02=ccode(sw02);
         sw02=sw02(12:length(sw02)-1)
         sw02=strrep(sw02,'~','');
         % Remplace les vitesses articulaires
         for c=1:M
            sw00=strrep(sw00,['Tp',num2str(c)],['u[',num2str(c-1),']']);
            sw01=strrep(sw01,['Tp',num2str(c)],['u[',num2str(c-1),']']);
            sw02=strrep(sw02,['Tp',num2str(c)],['u[',num2str(c-1),']']);
            sw00=strrep(sw00,['Dp',num2str(c)],['u[',num2str(c-1),']']);
            sw01=strrep(sw01,['Dp',num2str(c)],['u[',num2str(c-1),']']);
            sw02=strrep(sw02,['Dp',num2str(c)],['u[',num2str(c-1),']']);
         end
```

```
         % Remplace les termes trigo
         longt=size(slistet);
         for c=1:longt(1)
            eval(['umt=''u[',num2str(M+c-1),']'';']); % Production de la
chaine représentant l'entrée pour un terme trigo

eval(['sw00=strrep(sw00,''',deblank(slistet(c,:)),''',''',umt,''');']);

eval(['sw01=strrep(sw01,''',deblank(slistet(c,:)),''',''',umt,''');']);

eval(['sw02=strrep(sw02,''',deblank(slistet(c,:)),''',''',umt,''');']);
         end
         % Remplace les lamda
         sw00=strrep(sw00,'lamda',['u[',num2str(M+longt(1)),']']);
         sw01=strrep(sw01,'lamda',['u[',num2str(M+longt(1)),']']);
         sw02=strrep(sw02,'lamda',['u[',num2str(M+longt(1)),']']);
         % Écriture du résultat dans une variable unique
         eval(['sw00',num2str(a),num2str(b),'=sw00']);
         eval(['sw01',num2str(a),num2str(b),'=sw01']);
         eval(['sw02',num2str(a),num2str(b),'=sw02']);
      end
   end

   % Écriture du code du calcul de la matrice W
****************************************

   h=fopen('calcul_w.c','w') % fichier destination
   h2=fopen('calcul_w_s.c','r') % fichier source

   % Écriture du début de calcul_w
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*1*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Définition des constantes
   longc=size(slistec);
   for a=1:longc(1)
      if (mod(nlistec(a),1)~=0) % pour savoir si on ajoute le .0 après
la valeur numérique
         fprintf(h,'%s\n',['#define ',deblank(slistec(a,:)),'
',num2str(nlistec(a))]);
      else
         fprintf(h,'%s\n',['#define ',deblank(slistec(a,:)),'
',num2str(nlistec(a)),'.0']);
      end
   end
   fprintf(h,'%s\n',''); % espace

   % Écriture partielle de calcul_w
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*2*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
```

```
end

% Nombre d'entrées et de sorties
nent=M+1+longt(1); % calcul du nombre d'entrées
nsor=3*M*M; % calcul du nombre de sorties
fprintf(h,'%s\n',['    ssSetNumInputs(S,',num2str(nent),');']);
fprintf(h,'%s',['    ssSetNumOutputs(S,',num2str(nsor),');']);

% Écriture partielle de calcul_w
temp2=fscanf(h2,'%c',5);
while (~strcmp(temp2,'/*3*/'))
   fprintf(h,'%c',temp2(1));
   temp=fscanf(h2,'%c',1);
   temp2=[temp2(2:5),temp];
end

% Algorithme
% (Équations des matrices W00 et W01)
for a=1:M
   for b=1:M
      eval(['sw00=[sw00',num2str(a),num2str(b),','';''];']);
      eval(['temp=''y[',num2str((a-1)*M+b-1),']=',sw00,''';']);
      fprintf(h,'%s\n',temp);
   end
end
for a=1:M
   for b=1:M
      eval(['sw01=[sw01',num2str(a),num2str(b),','';''];']);
      eval(['temp=''y[',num2str((a-1)*M+b+8),']=',sw01,''';']);
      fprintf(h,'%s\n',temp);
   end
end
for a=1:M
   for b=1:M
      eval(['sw02=[sw02',num2str(a),num2str(b),','';''];']);
      eval(['temp=''y[',num2str((a-1)*M+b+17),']=',sw02,''';']);
      fprintf(h,'%s\n',temp);
   end
end

% Écriture de la fin de calcul_w
temp=fscanf(h2,'%c',1);
while (~strcmp(temp,''))
   fprintf(h,'%c',temp);
   temp=fscanf(h2,'%c',1);
end

fclose(h2);
fclose(h);
% fin de l'écriture du code du calcul de W


% Calcul de Y et a'
%%%
```

```
   temp='[';
   temp2='[';
   temp3='[';
   temp4='[';
   temp5='[';
   temp6='[';
   temp7='[';
   for a=1:M
      temp=[temp,'qrp',num2str(a),';'];
      eval(['syms qrp',num2str(a),' real']);
      temp2=[temp2,'qrpp',num2str(a),';'];
      eval(['syms qrpp',num2str(a),' real']);
      temp3=[temp3,'s',num2str(a),';'];
      eval(['syms s',num2str(a),' real']);
      temp4=[temp4,'e',num2str(a),';'];
      eval(['syms e',num2str(a),' real']);
      for b=1:M
         temp5=[temp5,'p_',num2str((a-1)*M+b),','];
         temp6=[temp6,'w',num2str((a-1)*M+b),','];
         temp7=[temp7,'r',num2str((a-1)*M+b),','];
         eval(['syms p_',num2str((a-1)*M+b),' real']);
         eval(['syms w',num2str((a-1)*M+b),' real']);
         eval(['syms r',num2str((a-1)*M+b),' real']);
      end
      temp5=[temp5(1:length(temp5)-1),';']; %enlève virgule de trop et
ajoute un ;
      temp6=[temp6(1:length(temp6)-1),';']; %enlève virgule de trop et
ajoute un ;
      temp7=[temp7(1:length(temp7)-1),';']; %enlève virgule de trop et
ajoute un ;
   end
   temp=[temp(1:length(temp)-1),']'];
   eval(['vectqrp=',temp,';']);
   temp2=[temp2(1:length(temp2)-1),']'];
   eval(['vectqrpp=',temp2,';']);
   temp3=[temp3(1:length(temp3)-1),']'];
   eval(['vects=',temp3,';']);
   temp4=[temp4(1:length(temp4)-1),']'];
   eval(['vecte=',temp4,';']);
   temp5=[temp5(1:length(temp5)-1),']'];
   eval(['matp=',temp5,';']);
   temp6=[temp6(1:length(temp6)-1),']'];
   eval(['matw=',temp6,';']);
   temp7=[temp7(1:length(temp7)-1),']'];
   eval(['matr=',temp7,';']);

   % Calcul de la matrice Y
   Tt=MTheta*vectqrpp+matc*vectqrp+gTheta;
   maty=jacobian(Tt,vectM);
   maty=simple(maty);

   % Calcul de la loi d'adaptation (da/dt)
   dasdt=matp*(maty.'*vects+matw.'*matr*vecte); % matp est -matp en
réalité (dans le schéma simulink)
   dasdt=simple(dasdt);

   % Remplacement des variables par des entrées
```

```
   for a=1:M
      % Transformation des variables symboliques en chaines de
caractères
      eval(['sdasdt=dasdt(a);']);
      sdasdt=strvcat(sdasdt);
      % Remplace les vitesses articulaires
????????????????????????????????????????????????????????????????????
      for b=1:M
         sdasdt=strrep(sdasdt,['Tp',num2str(b)],['u[',num2str(b-
1),']']);
         sdasdt=strrep(sdasdt,['Dp',num2str(b)],['u[',num2str(b-
1),']']);
      end
      % Remplace les e
      for b=1:M
         sdasdt=strrep(sdasdt,['e',num2str(b)],['u[',num2str(b-
1+M),']']);
      end
      % Remplace les s
      for b=1:M
         sdasdt=strrep(sdasdt,['s',num2str(b)],['u[',num2str(b-
1+2*M),']']);
      end
      % Remplace les qrp
      for b=1:M
         sdasdt=strrep(sdasdt,['qrp',num2str(b)],['u[',num2str(b-
1+3*M),']']);
      end
      % Remplace les qrpp
      for b=1:M
         sdasdt=strrep(sdasdt,['qrpp',num2str(b)],['u[',num2str(b-
1+4*M),']']);
      end
      % Remplace la matrice W
      for b=1:M*M
         sdasdt=strrep(sdasdt,['w',num2str(b)],['u[',num2str(b-
1+5*M),']']);
      end
      % Remplace la matrice P
      for b=1:M*M
         sdasdt=strrep(sdasdt,['p_',num2str(b)],['u[',num2str(b-
1+8*M),']']);
      end
      % Remplace la matrice R
      for b=1:M*M
         sdasdt=strrep(sdasdt,['r',num2str(b)],['u[',num2str(b-
1+11*M),']']);
      end
      % Remplace les termes trigo
      longt=size(slistet);
      for b=1:longt(1)
         umt=['u[',num2str(14*M+b-1),']']; % Production de la chaine
représentant l'entrée pour un terme trigo

eval(['sdasdt=strrep(sdasdt,''',deblank(slistet(b,:)),''',''',umt,''');'
]);
      end
```

```
      % Écriture du résultat dans une variable unique
      eval(['sdasdt',num2str(a),'=sdasdt']);
   end

   % Écriture du code du calcul de da/dt
*****************************************

   h=fopen('calcul_y_a.c','w') % fichier destination
   h2=fopen('calcul_y_a_s.c','r') % fichier source

   % Écriture du début de calcul_y_a
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*1*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Définition des constantes
   longc=size(slistec);
   for a=1:longc(1)
      if (mod(nlistec(a),1)~=0) % pour savoir si on ajoute le .0 après
la valeur numérique
         fprintf(h,'%s\n',['#define ',deblank(slistec(a,:)),'
',num2str(nlistec(a))]);
      else
         fprintf(h,'%s\n',['#define ',deblank(slistec(a,:)),'
',num2str(nlistec(a)),'.0']);
      end
   end
   fprintf(h,'%s\n',''); % espace

   % Écriture partielle de calcul_y_a
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*2*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Nombre d'entrées et de sorties
   nent=14*M+longt(1); % calcul du nombre d'entrées
   nsor=M; % calcul du nombre de sorties
   fprintf(h,'%s\n',['    ssSetNumInputs(S,',num2str(nent),');']);
   fprintf(h,'%s',['    ssSetNumOutputs(S,',num2str(nsor),');']);

   % Écriture partielle de calcul_y_a
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*3*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Algorithme
   % (Équations de da/dt)
```

```
   for a=1:M
      eval(['sdasdt=sdasdt',num2str(a),';']);
      temp=['y[',num2str(a-1),']=',sdasdt,';'];
      fprintf(h,'%s\n',temp);
   end

   % Écriture de la fin de calcul_y_a
   temp=fscanf(h2,'%c',1);
   while (~strcmp(temp,''))
      fprintf(h,'%c',temp);
      temp=fscanf(h2,'%c',1);
   end

   fclose(h2);
   fclose(h);
   % fin de l'écriture du code du calcul de Y et a


   % Calcul de la loi de commande (FF + PD)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Calcul de la loi de commande (anticipation + PD)
   loi=maty*vectM;
   loi=simple(loi);

   % Remplacement des variables par des entrées
   for a=1:M
      % Transformation des variables symboliques en chaines de
caractères
      eval(['sloi=loi(a);']);
      sloi=strvcat(sloi);
      % Remplace les a (masses identifiées)
      for b=1:M
         sloi=strrep(sloi,['M',num2str(b-1)],['u[',num2str(b-1),']']);
      end
      % Remplace les qrp
      for b=1:M
         sloi=strrep(sloi,['qrp',num2str(b)],['u[',num2str(b-1+M),']']);
      end
      % Remplace les qrpp (en réalité, qrpp-s(delta))
      for b=1:M
         sloi=strrep(sloi,['qrpp',num2str(b)],['u[',num2str(b-
1+2*M),']']);
      end
      % Remplace les vitesses
articulaires**********************************************erreur?
mélangé qp et qrp????
      for b=1:M
         sloi=strrep(sloi,['Tp',num2str(b)],['u[',num2str(b-
1+3*M),']']);
         sloi=strrep(sloi,['Dp',num2str(b)],['u[',num2str(b-
1+3*M),']']);
      end
      % Remplace les termes trigo
      longt=size(slistet);
      for b=1:longt(1)
```

```
         umt=['u[',num2str(4*M+b-1),']']; % Production de la chaine
représentant l'entrée pour un terme trigo

eval(['sloi=strrep(sloi,''',deblank(slistet(b,:)),''',''',umt,''');']);
      end

      % Écriture du résultat dans une variable unique
      eval(['sloi',num2str(a),'=sloi']);
   end

   % Écriture du code du calcul de la loi de commande
****************************************

   h=fopen('loi_commande.c','w') % fichier destination
   h2=fopen('loi_commande_s.c','r') % fichier source

   % Écriture du début de loi_commande
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*1*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Définition des constantes
   longc=size(slistec);
   for a=1:longc(1)
      if (mod(nlistec(a),1)~=0) % pour savoir si on ajoute le .0 après
la valeur numérique
         fprintf(h,'%s\n',['#define ',deblank(slistec(a,:)),'
',num2str(nlistec(a))]);
      else
         fprintf(h,'%s\n',['#define ',deblank(slistec(a,:)),'
',num2str(nlistec(a)),'.0']);
      end
   end
   fprintf(h,'%s\n',''); % espace

   % Écriture partielle de loi_commande
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*2*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Nombre d'entrées et de sorties
   nent=4*M+longt(1); % calcul du nombre d'entrées
   nsor=M; % calcul du nombre de sorties
   fprintf(h,'%s\n',['    ssSetNumInputs(S,',num2str(nent),');']);
   fprintf(h,'%s',['    ssSetNumOutputs(S,',num2str(nsor),');']);

   % Écriture partielle de loi_commande
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*3*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
```

```
      temp2=[temp2(2:5),temp];
   end

   % Algorithme
   % loi de commande
   for a=1:M
      eval(['sloi=sloi',num2str(a),';']);
      temp=['y[',num2str(a-1),']=',sloi,';'];
      fprintf(h,'%s\n',temp);
   end

   % Écriture de la fin de loi_commande
   temp=fscanf(h2,'%c',1);
   while (~strcmp(temp,''))
      fprintf(h,'%c',temp);
      temp=fscanf(h2,'%c',1);
   end

   fclose(h2);
   fclose(h);
   % fin de l'écriture du code du calcul de la loi de commande



   % Vecteur sin cos (slistet = vecteur de chaines contenant les
opérations trigo à inclure)

   % Écriture du code du calcul des termes trigo
*****************************************

   h=fopen('vect_sincos.c','w') % fichier destination
   h2=fopen('vect_sincos_s.c','r') % fichier source

   % Écriture du début de vect_sincos
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*1*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Nombre d'entrées et de sorties
   nent=M; % calcul du nombre d'entrées
   nsor=longt(1); % calcul du nombre de sorties
   fprintf(h,'%s\n',['    ssSetNumInputs(S,',num2str(nent),');']);
   fprintf(h,'%s',['    ssSetNumOutputs(S,',num2str(nsor),');']);

   % Écriture partielle de loi_commande
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*2*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Algorithme
   % Calcul des termes trigonométriques
```

```
   for a=1:longt(1)
      temp=deblank(slistet(a,:));
      temp=[temp,')'];
      for b=1:M
         temp=strrep(temp,'p','+u[');
         temp=strrep(temp,'m','-u[');
      end
      % Met des "]" après chaque entrée...
      ltemp=length(temp);
      b=2;
      while(b<=ltemp)
         if ((~isempty(str2num(temp(b-1))))&(isempty(str2num(temp(b)))))
            temp=[temp(1:b-1),']',temp(b:ltemp)];
            ltemp=length(temp);
         end
         b=b+1;
      end
      temp=strrep(temp,'s','sin(');
      temp=strrep(temp,'c','cos(');

      % Décrément de 1 des indices des theta (T1 -> u[0])
     % if (M>=10)
     %    for b=10:M
     %       temp=strrep(temp,num2str(b),num2str(b-1));
     %    end
     % end %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
problème si M>=10 !!!!!!!!!!!!
      for b=1:min(M,9)
         temp=strrep(temp,num2str(b),num2str(b-1));
      end

      temp=['y[',num2str(a-1),']=',temp,';'];
      fprintf(h,'%s\n',temp);
   end

   % Écriture de la fin de vect_sincos
   temp=fscanf(h2,'%c',1);
   while (~strcmp(temp,''))
      fprintf(h,'%c',temp);
      temp=fscanf(h2,'%c',1);
   end

   fclose(h2);
   fclose(h);
   % fin de l'écriture du code de calcul des termes trigo


   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Création des variables ctes_m et ctes_vg %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Construction du vecteur de constantes pour le calcul de la matrice
des masses
   % et du vecteur de termes centrifuges, de Coriolis et de gravité
   % 1-M : smasse2 (vecteur des masses)
   % M+1 - M+1+longt(1) : slistet (termes de trigo)
   ctes='[';
```

```
   longm=size(smasse2);
   for a=1:longm(1)
      ctes=[ctes,deblank(smasse2(a,:)),';'];
   end
   ctes=[ctes(1:length(ctes)-1),']']; %enlève le dernier ";" et ajoute
un ]
   save constantes_masses ctes


   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Compilation de la routine de résolution d'équation %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   eval(['!perl libconst.pl ',num2str(M)]);
   mex divmat.c

%%%%%%%%%%%%%%%%%%%
```

```
      end
   end
   Ndeg=compteur; % nombre de degrés de liberté
   jaco=jacot;

   if (M<Ndeg)
      temp=jaco.'*jaco;
      for a=1:M
         for b=1:M
            temp(a,b)=simple(temp(a,b));
         end
      end
      disp('jaco inv ')
      temp=inv(temp);
      for a=1:M
         for b=1:M
            temp(a,b)=simple(temp(a,b));
            disp('... ... ...')
         end
      end
      jig=temp*jaco.';
      for a=1:M %(non testé)
         for b=1:Ndeg
            jig(a,b)=simple(jig(a,b));
            disp('... ... ...')
         end
      end

   elseif (M==Ndeg)
      disp('jaco inv ')
      jig=inv(jaco);
      for a=1:M
         for b=1:M
            jig(a,b)=simple(jig(a,b));
            disp('... ... ...')
         end
      end

   else
      jig=jaco*jaco.'*inv(jaco*jaco.');
   end

   % Calcul de la vitesse articulaire en fonction de la vitesse
cartésienne
   % et construction du vecteur d'accélération articulaire
   syms xp yp zp Txp Typ Tzp real
   syms xpp ypp zpp Txpp Typp Tzpp real
   vectXp='[';
   vectXpp='[';
   for a=1:length(vectp)
      if (vectp(a)==1)
         vectXp=[vectXp,'xp;'];
         vectXpp=[vectXpp,'xpp;'];
      elseif (vectp(a)==2)
         vectXp=[vectXp,'yp;'];
         vectXpp=[vectXpp,'ypp;'];
      elseif (vectp(a)==3)
```

```
            vectXp=[vectXp,'zp;'];
            vectXpp=[vectXpp,'zpp;'];
        elseif (vectp(a)==4)
            vectXp=[vectXp,'Txp;'];
            vectXpp=[vectXpp,'Txpp;'];
        elseif (vectp(a)==5)
            vectXp=[vectXp,'Typ;'];
            vectXpp=[vectXpp,'Typp;'];
        elseif (vectp(a)==6)
            vectXp=[vectXp,'Tzp;'];
            vectXpp=[vectXpp,'Tzpp;'];
        end
    end
    vectXp=vectXp(1:(length(vectXp)-1)); % enlève le dernier ;
    vectXp=[vectXp,']']
    vitart=jig*vectXp;


    vectXpp=vectXpp(1:(length(vectXpp)-1)); % enlève le dernier ;
    vectXpp=[vectXpp,']'];

    % Construction du vecteur de position articulaire
    vectq=sym(zeros(M,1));
    for a=1:M
        if (ki(a)==1)
            eval(['syms T',num2str(a),' real']);
            eval(['vectq(a)=T',num2str(a)]);
        else
            eval(['syms d',num2str(a),' real']);
            eval(['vectq(a)=d',num2str(a)]);
        end
    end

    % Calcul de la dérivée de l'inverse du Jacobien de vitesse
    disp('jaco dérivé')
    % dérivée élément par élément pour éviter des trop grosses opérations
pour Matlab
    jigd=sym(zeros(M,Ndeg));
    for a=1:M
        for b=1:Ndeg
            for c=1:M %dérivée en chaine...
                jigd(a,b)=jigd(a,b)+diff(jig(a,b),vectq(c))*vectqp(c)
                disp('...')
            end
        end
    end
    disp('acc art')
    % Calcul de l'accélération articulaire en fonction des vitesse et
accélération cartésienne
    accart=jigd*vectXp+jig*vectXpp;

    % Remplacement des variables par des entrées pour l'écriture du code
*******************
    disp('remplacement')
    for a=1:M
        svitart=strvcat(vpa(vitart(a)));
        saccart=strvcat(vpa(accart(a)));
```

```
      % Remplacement des vitesses et accélérations cartésiennes
      for b=Ndeg:-1:1
         eval(['sXp=',vectXp,';']);
         sXp=strvcat(sXp(b));
         eval(['sXpp=',vectXpp,';']);
         sXpp=strvcat(sXpp(b));

         ut=['u[',num2str(b+Ndeg-1),']'];
         svitart=strrep(svitart,sXpp,ut);
         saccart=strrep(saccart,sXpp,ut);
         ut=['u[',num2str(b-1),']'];
         svitart=strrep(svitart,sXp,ut);
         saccart=strrep(saccart,sXp,ut);
      end
      % Remplacement des positions et vitesses articulaires
      for b=M:-1:1
         sq=strvcat(vectq(b));
         ut=['u[',num2str(b+2*Ndeg-1),']'];
         svitart=strrep(svitart,sq,ut);
         saccart=strrep(saccart,sq,ut);

         sqp=strvcat(vectqp(b));
         yt=['y[',num2str(b-1),']'];
         saccart=strrep(saccart,sqp,yt);
      end

      % Sauvegarde de la variable
      eval(['svitart',num2str(a),'=svitart']);
      eval(['saccart',num2str(a),'=saccart']);
   end

   % Enlève les puissances du Jacobien (avec fonction améliorée
supportant les parentaises...)
   listetermesJ='';
   exposantsJ='';
   for a=1:M

eval(['[svitart',num2str(a),',listetermesJ,exposantsJ]=enleve_puissances
2(svitart', ...
         num2str(a),',listetermesJ,exposantsJ);']);

eval(['[saccart',num2str(a),',listetermesJ,exposantsJ]=enleve_puissances
2(saccart', ...
         num2str(a),',listetermesJ,exposantsJ);']);
   end

   % Écriture du code de calcul de la vitesse et accélération
articulaires

   h=fopen('jacobien_vitacc.c','w') % fichier destination
   h2=fopen('jacobien_vitacc_s.c','r') % fichier source

   % Écriture du début de jacobien_vitacc
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*1*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
```

```
      temp2=[temp2(2:5),temp];
   end

   % Définition des constantes
   longc=size(slistec);
   for a=1:longc(1)
      if (mod(nlistec(a),1)~=0) % pour savoir si on ajoute le .0 après
la valeur numérique
         fprintf(h,'%s\n',['#define ',deblank(slistec(a,:)),'
',num2str(nlistec(a))]);
      else
         fprintf(h,'%s\n',['#define ',deblank(slistec(a,:)),'
',num2str(nlistec(a)),'.0']);
      end
   end
   fprintf(h,'%s\n',''); % espace

   % Écriture partielle de jacobien_vitacc
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*2*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Nombre d'entrées et de sorties
   nent=2*Ndeg+M; % calcul du nombre d'entrées
   nsor=2*M; % calcul du nombre de sorties
   fprintf(h,'%s\n',['    ssSetNumInputs(S,',num2str(nent),');']);
   fprintf(h,'%s',['    ssSetNumOutputs(S,',num2str(nsor),');']);

   % Écriture partielle de jacobien_vitacc
   temp2=fscanf(h2,'%c',5);
   while (~strcmp(temp2,'/*3*/'))
      fprintf(h,'%c',temp2(1));
      temp=fscanf(h2,'%c',1);
      temp2=[temp2(2:5),temp];
   end

   % Algorithme
   % Définition des variables temporaires (calcul des puissances)
   % listetermesJ : liste des termes exposés (char)
   % exposantsJ : exposants des termes (double)

   % Définition des variables
   temp='double ';
   lt=length(exposantsJ);
   for a=1:lt
      temp=[temp,'temp',num2str(a),', '];
   end
   temp=[temp(1:length(temp)-2),';'];
   fprintf(h,'%s\n',temp);
   fprintf(h,'%s\n',''); % espace

   % Écriture du calcul des puissances par multiplications répétées
   for a=1:lt
      temp=['temp',num2str(a),'=',deblank(listetermesJ(a,:)),';'];
```

```
      fprintf(h,'%s\n',temp);
      % Production de la chaine des "temp*temp*temp..."
      temp='';
      for b=1:exposantsJ(a)
         temp=[temp,'temp',num2str(a),'*'];
      end
      temp=temp(1:length(temp)-1); %enlève le * de trop
      temp=['temp',num2str(a),'=',temp,';'];
      fprintf(h,'%s\n',temp);
   end
   fprintf(h,'%s\n',''); % espace

   % Équations de calcul de la vitesse et de l'accélération articulaire
   for a=1:M
      eval(['svitart=svitart',num2str(a)]);
      temp=['y[',num2str(a-1),']=',strvcat(svitart),';'];
      fprintf(h,'%s\n',temp);
   end
   for a=1:M
      eval(['saccart=saccart',num2str(a)]);
      temp=['y[',num2str(a-1+M),']=',strvcat(saccart),';'];
      fprintf(h,'%s\n',temp);
   end

   % Écriture de la fin de jacobien_vitacc
   temp=fscanf(h2,'%c',1);
   while (~strcmp(temp,''))
      fprintf(h,'%c',temp);
      temp=fscanf(h2,'%c',1);
   end

   fclose(h2);
   fclose(h);
   % fin de l'écriture du code de calcul de la vitesse et accélération
articulaires

end

% Écriture du code de produit matrice-vecteur
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Création de la matrice et du vecteur en symbolique
temp1='vect=[';
temp2='mat=[';
for a=1:M
   eval(['syms v',num2str(a),' real']);
   eval(['temp1=[temp1,''v',num2str(a),';''];']);
   for b=1:M
      eval(['syms m',num2str(a),num2str(b),' real']);
      eval(['temp2=[temp2,''m',num2str(a),num2str(b),','''];']);
   end
   temp2=[temp2(1:length(temp2)-1),';'];
end
temp1=[temp1(1:length(temp1)-1),']'];
eval(temp1);
temp2=[temp2(1:length(temp2)-1),']'];
eval(temp2);
```

```
% Calcul...
sortie=mat*vect;
sortie=simple(sortie);

% Remplacement des variables par des entrées
for a=1:M
   ssortie=strvcat(sortie(a));
   for b=1:M
      ssortie=strrep(ssortie,['v',num2str(b)],['u[',num2str(M*M+b-
1),']']);
      for c=1:M

ssortie=strrep(ssortie,['m',num2str(b),num2str(c)],['u[',num2str((b-
1)*M+c-1),']']);
      end
   end
   eval(['ssortie',num2str(a),'=ssortie;'])
end

% Écriture du code de prodmatvectn
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

h=fopen('prodmatvectn.c','w') % fichier destination
h2=fopen('prodmatvectn_s.c','r') % fichier source

% Écriture du début de prodmatvectn
temp2=fscanf(h2,'%c',5);
while (~strcmp(temp2,'/*1*/'))
   fprintf(h,'%c',temp2(1));
   temp=fscanf(h2,'%c',1);
   temp2=[temp2(2:5),temp];
end

% Nombre d'entrées et de sorties
nent=M*M+M; % calcul du nombre d'entrées
nsor=M; % calcul du nombre de sorties
fprintf(h,'%s\n',['    ssSetNumInputs(S,',num2str(nent),');']);
fprintf(h,'%s',['    ssSetNumOutputs(S,',num2str(nsor),');']);

% Écriture partielle de prodmatvectn
temp2=fscanf(h2,'%c',5);
while (~strcmp(temp2,'/*2*/'))
   fprintf(h,'%c',temp2(1));
   temp=fscanf(h2,'%c',1);
   temp2=[temp2(2:5),temp];
end

% Algorithme
% Algorithme de calcul du produit matrice vecteur d'ordre n

for a=1:M
   eval(['ssortie=ssortie',num2str(a)]);
   temp=['y[',num2str(a-1),']=',strvcat(ssortie),';'];
   fprintf(h,'%s\n',temp);
end
```

```
% Écriture de la fin de jacobien_vitacc
temp=fscanf(h2,'%c',1);
while (~strcmp(temp,''))
   fprintf(h,'%c',temp);
   temp=fscanf(h2,'%c',1);
end

fclose(h2);
fclose(h);
```

Programme Matlab de remplacement des puissances par des multiplications répétées

```
function [vars]=enleve_puissances(vare);
% enleve_puissances.m
%
% Ce programme effectue un balayage d'une variable (scalaire, vecteur ou
matrice)
% et transforme les opérations de puissance (nombre entier) en
oopérations de
% multiplications répétées.
%
% entrées et sorties :
% 1) vers, vare : variable à modifier
%
% Par : Martin de Montigny
% Février 1999, UQTR

% Calcul de la dimension de la variable d'entrée
[nl,nc]=size(vare);

% Routine principale de recherche et modification de la variable
rendu=1; %Position courante du curseur de recherche de "^"
for a=1:nl
   for b=1:nc

      var=vare(a,b); % Extraction de l'élément à traiter
      svar=strvcat(var); % Transformation de l'élément à traiter en
chaine de caractères
      long=length(svar); % Longueur de la variable (en caractères)
      % Boucle de recherche des "^"
      c=1;
      while (c<=long)

         car=svar(c);
         if car=='^'

            renduv=c;
            while ((car~='+')&(car~='-
')&(car~='*')&(car~='/')&(car~='('))
               renduv=renduv-1;
               if renduv>1
                  car=svar(renduv-1);
```

```
               else
                  car='+';
               end
            end
            % "rendu" pointe maintenant le début de la variable à la
puissance x
            varpui=svar(renduv:(c-1)) % Capture de la variable à la
puissance x
            % Recherche de l'exposant
            rendue=c;

            car=svar(rendue);
            while ((car~='+')&(car~='-
')&(car~='*')&(car~='/')&(car~=')'))
               rendue=rendue+1;
               if rendue<long
                  car=svar(rendue+1);
               else
                  car='+';
               end
            end

            % "rendu" pointe maintenant la fin de la puissance x de la
variable
            pui=str2num(svar((c+1):rendue));

            % Construction de la variable multipliée par elle même "pui"
fois qui
            % remplacera la variable à la puissance "pui"
            varm=varpui; %initialisation de la nouvelle variable
            for d=2:pui
               varm=[varm,'*',varpui];
            end

            % Reconstitution de la variable (élément sous forme de
chaine de car.)
            svar=[svar(1:(renduv-1)),varm,svar((rendue+1):long)];

            % Ajustement de la nouvelle longueur et du nouveau c !!!
            long=long+length(varm)-length([varpui,'^',pui]);
            c=c+length(varm)-length([varpui,'^',pui]);

         end
         c=c+1;
      end

      % Transformation de l'élément de la variable modifié en symbolique
      vars(a,b)=sym(svar);

   end
end
```

Programme Matlab de remplacement des puissances par des multiplications répétées (version 2)

```
function
[expression,listetermes,exposants]=enleve_puissances2(expression,listete
rmes,exposants);
% enleve_puissances2.m
%
% Ce programme remplace les termes à une puissances (d'une expression
sous forme de
% chaine de caractères) par une variable.
%
% limites : -il doit y avoir une parentaise ")" avant tout exposant
%           -les exposants doivent être entiers et <=9
%           -le programme ne regarde pas si le terme exposé trouvé se
trouve déjà dans la liste
%
% Entrées : expression :   expression dont on doit enlever les exposants
(char)
%           listetermes :  vecteur de chaines de car. contenant les
termes trouvés (char)
%           exposants :    vecteur des exposants des termes trouvés
(double)
%
% Par : Martin de Montigny
% 1999, UQTR
% version 1 : 19 avril 1999
% mod 22 sept 99 : si sin ou cos avant la première parentaise...
%                  le programme vérifie s'il la nouvelle constante se
trouve déjà dans la liste

a=1;
t=expression;
le=length(exposants);
noterme=1+le; %numéro du prochain terme à remplacer
lt=length(expression);
while (a<=lt)
   if (t(a)=='^')
      if (t(a-1)==')') % si parentaise avant ^
         compteur=1; % compteur de parentaises
         b=a-2; % pointe le caractère avant le signe exposant ^
         while ((compteur>=1)&(b>=0))
            if (t(b)==')')
               compteur=compteur+1;
            elseif (t(b)=='(')
               compteur=compteur-1;
            end
            b=b-1;
         end
         % b pointe le car. avant la première (

         % ajustement de b si la parentaise appartient à un sin ou un
cos
         if ((t(b-2:b)=='sin')|(t(b-2:b)=='cos'))
            b=b-3;
```

```
         end

         % a pointe le car. après la dernière )

         % Capture du terme et de l'exposant
         terme=t(b+1:a-1);
         exposant=str2num(t(a+1));

         % vérification de la liste (termes + exposants) pour savoir si
la nouvelle
         % constante est déjà présente
         test=0;
         numero_exposant=noterme; %numéro de la nouvelle constante
         for c=1:length(exposants)
            if
(strcmp(deblank(listetermes(c,:)),terme)&(exposants(c)==exposant))
               test=1;
               numero_exposant=c; % numéro de la constante trouvée
(pareille à la nouvelle)
            end
         end

         if (test==0)
            listetermes=strvcat(listetermes,terme); %capture le terme
            exposants=[exposants;exposant]; % exposant sous forme
numérique
            noterme=noterme+1; % constante suivante...
         end
         t=[t(1:b),'temp',num2str(numero_exposant),t(a+2:lt)]; %
remplacement de terme par la variable
         lt=length(t); % mise à jour de la longueur de t
         a=a-length(terme)+length(num2str(numero_exposant))+4; % mise à
jour de "a" en fonction du nombre de caractères enlevés et ajoutés

      end
   end
   a=a+1;
end

% sauve l'expression
expression=t;
```

Programme Matlab de remplacement des termes trigonométriques par des constantes

```
function [vars,listes]=remplace_trigo(vare,listee);
% remplace_trigo.m
%
% Ce programme effectue un balayage d'une variable (scalaire, vecteur ou
matrice)
% et transforme les opérateurs de trigonométrie en constantes pré
calculées.
%
% entrées et sorties :
```

```
% 1) vers, vare : variable à modifier
% 2) listee, listes : liste des constantes trigonométriques à
précalculer (et dans cet ordre dans
%                     le vecteur du simulateur sur Simulink)
%
% Par : Martin de Montigny
% Février 1999, UQTR

% Calcul de la dimension de la variable d'entrée
[nl,nc]=size(vare);

% Initialisation de la nouvelle liste
listes=listee;

% Routine principale de recherche et modification de la variable
for a=1:nl
   for b=1:nc

      var=vare(a,b); % Extraction de l'élément à traiter
      svar=strvcat(var); % Transformation de l'élément à traiter en
chaine de caractères
      long=length(svar); % Longueur de la variable (en caractères)

      % Vecteurs de position des sin et cos possibles dans le vecteur
actuel (élément)
      vecposc=findstr(svar,'cos(');
      vecposs=findstr(svar,'sin(');

      % Calcul du nombre de sin et cos possibles dans la chaine de
caractères
      nsin=length(vecposs);
      ncos=length(vecposc);

      for c=1:nsin
         % Vérification de la validité du sin trouvé
         if ((vecposs(c)==1)|(svar(vecposs(c)-1)=='(')|(svar(vecposs(c)-
1)=='+')|(svar(vecposs(c)-1)=='-')|(svar(vecposs(c)-
1)=='*')|(svar(vecposs(c)-1)=='/'))

            % Capture du terme interne au sinus
            d1=vecposs(c)+4;
            d2=d1;
            while (svar(d2+1)~=')')
               d2=d2+1;
            end
            % d1 pointe le premier caractère du terme interne du sinus
            % d2 pointe le dernier caractère du terme interne du sinus
            % Capture du terme interne
            termeint=svar(d1:d2);

            % Analyse du terme interne pour connaître les angles
impliqués
            % Trouver la position des angles theta (ex : "T1+T2-T3" ->
[1 4 7])
            vecposa=findstr(termeint,'T');
            % Vérifie le signe de chaque angle theta
```

```
            % Angles : vecteur des angles trouvés (sous forme numérique,
double)
            angles=[];
            for d=1:length(vecposa)
               % Caractère avant le T actuel

               cara=svar(d1+vecposa(d)-2);
               % Recherche d'un angle additionné
               if ((cara=='(')|(cara=='+'))
                  % Recherche de la fin du nombre représentant le numéro
de l'angle
                  test=0;
                  e=0;
                  while test==0
                     e=e+1;
                     if ~length(str2num(svar(d1+vecposa(d)+e)))
                        test=1;
                     end
                  end
                  angle=svar(d1+vecposa(d):d1+vecposa(d)+e-1); % angle
sous forme de chaine
                  angles=[angles,str2num(angle)]; %angles sous forme
numérique
                  angles=sort(angles); % Mise en ordre du vecteur de
numéros d'angles (pour reconnaissance)
               elseif (cara=='-')
                  % Recherche de la fin du nombre représentant le numéro
de l'angle
                  test=0;
                  e=0;
                  while test==0
                     e=e+1;
                     if ~length(str2num(svar(d1+vecposa(d)+e)))
                        test=1;
                     end
                  end
                  angle=svar(d1+vecposa(d):d1+vecposa(d)+e-1); % angle
sous forme de chaine
                  angles=[angles,-str2num(angle)]; %angles sous forme
numérique
                  angles=sort(angles); % Mise en ordre du vecteur de
numéros d'angles (pour reconnaissance)
               end

            end

            % Construction de la constante qui remplacera l'opérateur
trigo
            constante='s';
            for e=1:length(angles)
               if angles(e)>=0
                  constante=[constante,'p',num2str(angles(e))];
               else
                  constante=[constante,'m',num2str(abs(angles(e)))];
               end
            end
            longnt=length(constante); %longueur du nouveau terme
```

```
            % Vérification de la présence de la nouvelle constante dans
la liste
            % et ajout si elle n'y est pas
            [ncliste,nlliste]=size(listes); % ncliste=nombre d'éléments
déjà présents dans la liste
            test=0;
            for e=1:ncliste
               if strcmp(deblank(listes(e,:)),constante)
                  test=1;
               end
            end
            if (test==0)
               listes=strvcat(listes,constante);
            end

            % Reconstitution de la variable (élément sous forme de
chaine de car.)
            svar=[svar(1:d1-5),constante,svar(d2+2:long)];

            % Correction des paramètres de "svar" car la longueur a
changé...
            long=length(svar); % Longueur de la variable (en caractères)
            % Vecteurs de position des sin et cos possibles dans le
vecteur actuel (élément)
            % !!!!!! Ajustement seulement pour les termes à la suite du
terme modifié
            vecposc=vecposc+(vecposc>d1)*(-(d2-d1+6)+longnt);
            vecposs=vecposs+(vecposs>d1)*(-(d2-d1+6)+longnt);


         end % if

      end % c

      for c=1:ncos
         % Vérification de la validité du cos trouvé
         if ((vecposc(c)==1)|(svar(vecposc(c)-1)=='(')|(svar(vecposc(c)-
1)=='+')|(svar(vecposc(c)-1)=='-')|(svar(vecposc(c)-
1)=='*')|(svar(vecposc(c)-1)=='/'))

            % Capture du terme interne au sinus
            d1=vecposc(c)+4;
            d2=d1;
            while (svar(d2+1)~=')')
               d2=d2+1;
            end
            % d1 pointe le premier caractère du terme interne du cosinus
            % d2 pointe le dernier caractère du terme interne du cosinus
            % Capture du terme interne
            termeint=svar(d1:d2);

            % Analyse du terme interne pour connaître les angles
impliqués
            % Trouver la position des angles theta (ex : "T1+T2-T3" ->
[1 4 7])
            vecposa=findstr(termeint,'T');
```

```
            % Vérifie le signe de chaque angle theta
            % Angles : vecteur des angles trouvés (sous forme numérique,
double)
            angles=[];
            for d=1:length(vecposa)
               % Caractère avant le T actuel
               cara=svar(d1+vecposa(d)-2);
               % Recherche d'un angle additionné
               if ((cara=='(')|(cara=='+'))
                  % Recherche de la fin du nombre représentant le numéro
de l'angle
                  test=0;
                  e=0;
                  while test==0
                     e=e+1;
                     if ~length(str2num(svar(d1+vecposa(d)+e)))
                        test=1;
                     end
                  end
                  angle=svar(d1+vecposa(d):d1+vecposa(d)+e-1); % angle
sous forme de chaine
                  angles=[angles,str2num(angle)]; %angles sous forme
numérique
                  angles=sort(angles); % Mise en ordre du vecteur de
numéros d'angles (pour reconnaissance)
               elseif (cara=='-')
                  % Recherche de la fin du nombre représentant le numéro
de l'angle
                  test=0;
                  e=0;
                  while test==0
                     e=e+1;
                     if ~length(str2num(svar(d1+vecposa(d)+e)))
                        test=1;
                     end
                  end
                  angle=svar(d1+vecposa(d):d1+vecposa(d)+e-1); % angle
sous forme de chaine
                  angles=[angles,-str2num(angle)]; %angles sous forme
numérique
                  angles=sort(angles); % Mise en ordre du vecteur de
numéros d'angles (pour reconnaissance)
               end

            end

            % Construction de la constante qui remplacera l'opérateur
trigo
            constante='c';
            for e=1:length(angles)
               if angles(e)>=0
                  constante=[constante,'p',num2str(angles(e))];
               else
                  constante=[constante,'m',num2str(abs(angles(e)))];
               end
            end
            longnt=length(constante); %longueur du nouveau terme
```

```
            % Vérification de la présence de la nouvelle constante dans
la liste
            % et ajout si elle n'y est pas
            [ncliste,nlliste]=size(listes); % ncliste=nombre d'éléments
déjà présents dans la liste
            test=0;
            for e=1:ncliste
               if strcmp(deblank(listes(e,:)),constante)
                  test=1;
               end
            end
            if (test==0)
               listes=strvcat(listes,constante);
            end

            % Reconstitution de la variable (élément sous forme de
chaine de car.)
            svar=[svar(1:d1-5),constante,svar(d2+2:long)];

            % Correction des paramètres de "svar" car la longueur a
changé...
            long=length(svar); % Longueur de la variable (en caractères)
            % Vecteurs de position des sin et cos possibles dans le
vecteur actuel (élément)
            vecposc=vecposc-(d2-d1+6)+longnt;
            vecposs=vecposs-(d2-d1+6)+longnt;


         end % if

      end % c

      % Transformation de l'élément de la variable modifié en symbolique
      vars(a,b)=sym(svar);

   end % b
end % a
```

Programme Matlab de simplification des constantes

```
function [vsor,lcon,vcon]=remplace_constante(vent,lcon,vcon,fichier);
% remplace_constante.m
%
% Ce programme regroupe les constante après avoir effecté un "expand"
% sur la variable d'entrée.  Les constantes regroupées sont transformées
% en constantes créées sous la forme c0,c1,c2...
%
% Les constantes à regrouper sont sous la forme :
% l0, l1, l2...
% L0, L1, L2...
% t111, t112, t113...
%
```

```
% NOTE : IL NE DOIT PAS Y AVOIR DE PARANTAISES !!!
% FAIRE UN EXPAND, ENSUITE, ENLEVER LES PUISSANCES, LA TRIGO ET EXÉCUTER
CE PROGRAMME
% NOTE 2 : NE PAS FAIRE PASSER UNE MATRICE 2 FOIS PAR CE PROGRAMME !
(POUR L'INSTANT)
% NOTE 3 : PROBLÈME ACTUEL : IL NE DOIT PAS Y AVOIR DE DIVISION PAR UNE
LONGUEUR OU UN
%          ÉLÉMENT D'UN TENSEUR (POUR LE PERMETTRE, ON DEVRA AJOUTER UNE
SECTION...)
%
% L'entrée est : vent : variable d'entrée
%                lcon : liste des constantes présentes (string)
%                vcon : valeur des constantes présentes (double)
%                fichier : nom du fichier .mat des paramètres numériques
du robot
%
% Les sorties sont : vsor : variable de sortie
%                    lcon : liste des constantes créées (string)
%                    vcon : valeur des constantes créées (double)
%
% Par : Martin de Montigny
% Février 1999
% UQTR
%
% Modifications :
% 8 avril 1999 : Élargi le champ d'action du programme.  Avant, il
remplaçait les constantes multiples seulement.
%                Maintenant, il remplace les constantes uniques ET
multiples de façon à pouvoir fournir la liste
%                complète des constantes à utiliser lors de la
simulation (changement au niveau du "if" de ncons.)

% Détermination de la dimension de la variable à traiter
[nligne,ncolonne]=size(vent);

% Traitement de la variable
for a=1:nligne
   for b=1:ncolonne

      % Mise de l'élément à traiter sous forme de chaîne de caractères
      svar=strvcat(vent(a,b));
      long=length(svar); % longueur actuelle de l'élément total

      % Tant qu'il reste des termes, les capturer un par un et remplacer
les constantes par une constante précalculée
      pos=1; % position actuelle du curseur chercheur de termes

      while (pos<=long)

         % Test pour savoir si le curseur "pos" est rendu au premier
caractère d'un terme
         if ((svar(pos)~='-')&((pos==1)|(svar(pos-1)=='+')|(svar(pos-
1)=='-')))
            debut=pos;
            c=pos;

            while ((c<long)&((svar(c+1)~='+')&(svar(c+1)~='-')))
```

```
            c=c+1;
         end
         % "c" pointe le dernier caractère du terme trouvé
         fin=c;
         terme=svar(debut:fin); % Capture du terme
         ltermea=length(terme); % Longueur du terme présent (pour
test de modif. de svar à la fin)
         lterme=length(terme);

         % Initialisation des vecteurs de constantes trouvables dans
le terme
         cl=[];
         cL=[];
         ct=[];
         nl=findstr(terme,'l');
         nL=findstr(terme,'L');
         nt=findstr(terme,'t');
         for c=1:length(nl)
            % (Si un opérateur (-+*) est présent avant la constante
ou si elle est au début du terme)
            % et si le caractère après la constante est un chiffre

            if (((nl(c)==1)|(terme(nl(c)-1)=='-')|(terme(nl(c)-
1)=='+')|(terme(nl(c)-1)=='*'))&(length(str2num(terme(nl(c)+1)))))
               % Capture du nombre de la constante
               d=1;
               snombre='';
               while
(((nl(c)+d)<=lterme)&(length(str2num(terme(nl(c)+d)))))
                  snombre=[snombre,terme(nl(c)+d)];
                  d=d+1;
               end
               nombre=str2num(snombre); % numéro de la constante "l"
               cl=sort([cl,nombre]);
            end
         end
         for c=1:length(nL)
            % (Si un opérateur (-+*) est présent avant la constante
ou si elle est au début du terme)
            % et si le caractère après la constante est un chiffre
            if (((nL(c)==1)|(terme(nL(c)-1)=='-')|(terme(nL(c)-
1)=='+')|(terme(nL(c)-1)=='*'))&(length(str2num(terme(nL(c)+1)))))
               % Capture du nombre de la constante
               d=1;
               snombre='';
               while
(((nL(c)+d)<=lterme)&(length(str2num(terme(nL(c)+d)))))
                  snombre=[snombre,terme(nL(c)+d)];
                  d=d+1;
               end
               nombre=str2num(snombre); % numéro de la constante "L"
               cL=sort([cL,nombre]);
            end
         end
         for c=1:length(nt)
            % (Si un opérateur (-+*) est présent avant la constante
ou si elle est au début du terme)
```

```
            % et si le caractère après la constante est un chiffre
            if (((nt(c)==1)|(terme(nt(c)-1)=='-')|(terme(nt(c)-
1)=='+')|(terme(nt(c)-1)=='*'))&(length(str2num(terme(nt(c)+1)))))
               % Capture du nombre de la constante
               d=1;
               snombre='';
               while
(((nt(c)+d)<=lterme)&(length(str2num(terme(nt(c)+d)))))
                  snombre=[snombre,terme(nt(c)+d)];
                  d=d+1;
               end
               nombre=str2num(snombre); % numéro de la constante "t"
               ct=sort([ct,nombre]);
            end
         end

         % Constitution de la constante équivalente, si besoin est et
remplacement dans le terme
         lnl=length(cl); %longueur de cl et non de nl pour ne pas
inclure les constantes divisantes...
         lnL=length(cL);
         lnt=length(ct);
%          if
~(((lnl==1)&(lnL==0)&(lnt==0))|((lnl==0)&(lnL==1)&(lnt==0))|((lnl==0)&(l
nL==0)&(lnt==1))|((lnl==0)&(lnL==0)&(lnt==0)))
            if ~((lnl==0)&(lnL==0)&(lnt==0))

            % Lors de la constitution de la nouvelle constante, passe
les termes de division (après un "/")
            ncons='';
            for c=1:lnl

               if ((nl(c)==1)|(terme(nl(c)-1)~='/'))
                  ncons=[ncons,'l',num2str(cl(c))];
               end
            end
            for c=1:lnL
               if ((nL(c)==1)|(terme(nL(c)-1)~='/'))
                  ncons=[ncons,'L',num2str(cL(c))];
               end
            end
            for c=1:lnt
               if ((nt(c)==1)|(terme(nt(c)-1)~='/'))
                  ncons=[ncons,'t',num2str(ct(c))];
               end
            end

            % Vérifie si la constante créée existe déjà dans la liste
            % Si oui, ne rien faire.  Sinon, ajouter la constante à
la liste de
            % même que sa valeur numérique.
            nombcons=length(vcon);
            test=0;
            for c=1:nombcons
               if (strcmp(deblank(lcon(c,:)),ncons))
                  test=1;
               end
```

```
                end
                if (test==0)
                   save temp
                   eval(['load ',fichier]);
                   total=1; % Initialisation de la valeur du produit des
constantes
                   if (lnl>0)
                      for c=1:lnl
                         eval(['total=total*l',num2str(cl(c)),';']);
                      end
                   end
                   if (lnL>0)
                      for c=1:lnL
                         eval(['total=total*L',num2str(cL(c)),';']);
                      end
                   end
                   if (lnt>0)
                      for c=1:lnt
                         eval(['total=total*t',num2str(ct(c)),';']);
                      end
                   end

                   lcon2=strvcat(lcon,ncons);
                   vcon2=[vcon;total];
                   load temp
                   lcon=lcon2;
                   vcon=vcon2;
                   clear lcon2 vcon2
                end

                % Modifier le terme (effacer les anciennes constantes et
écrire la nouvelle)
                c=1;
                lterme=length(terme);
                while (c<=lterme)

                   if ((terme(c)=='l')&((c==1)|(terme(c-1)~='/')))
                      % Recherche de la longueur du nombre du "l" à
effacer
                      d=1;
                      while
(((c+d)<=lterme)&(length(str2num(terme(c+d)))))
                         d=d+1;
                      end
                      % c+d pointe le caractère après ancienne constante
                      % Effaçage du "l"
                      %(si le caractère d'avant l'ancienne constante est
un *, alors l'enlever)
                      if ((c==1)|(terme(c-1)=='*'))
                         if (c~=1) % test c==1 pour éviter erreur si c=1
pour terme(c-1)...
                            enlevemul=-1;
                         end
                      else
                         enlevemul=0;
                      end
                      if (c==1)
```

```
                    terme=terme(c+d:lterme);
                elseif (c+d>=lterme)
                    terme=terme(1:(c-1)+enlevemul);
                else
                    terme=[terme(1:(c-
1)+enlevemul),terme(c+d:lterme)];
                end
                lterme=length(terme); % Longueur du nouveau terme
                c=c-1; % Si on enlève un terme, un regarde encore à
la même place pour savoir s'il y a une autre constante

            end

            c=c+1;
        end

        c=1;
        lterme=length(terme);
        while (c<=lterme)

            if ((terme(c)=='L')&((c==1)|(terme(c-1)~='/')))
                % Recherche de la longueur du nombre du "L" à
effacer
                d=1;
                while
(((c+d)<=lterme)&(length(str2num(terme(c+d)))))
                    d=d+1;
                end
                % d pointe le caractère après ancienne constante
                % Effaçage du "L"
                %(si le caractère d'avant l'ancienne constante est
un *, alors l'enlever)
                if ((c==1)|(terme(c-1)=='*'))
                    if (c~=1) % test c==1 pour éviter erreur si c=1
pour terme(c-1)...
                        enlevemul=-1;
                    end
                else
                    enlevemul=0;
                end
                if (c==1)
                    terme=terme(c+d:lterme);
                elseif (c+d>=lterme)
                    terme=terme(1:(c-1)+enlevemul);
                else
                    terme=[terme(1:(c-
1)+enlevemul),terme(c+d:lterme)];
                end
                lterme=length(terme); % Longueur du nouveau terme
                c=c-1; % Si on enlève un terme, un regarde encore à
la même place pour savoir s'il y a une autre constante

            end

            c=c+1;
        end
```

```
                c=1;
                lterme=length(terme);
                while (c<=lterme)

                   if ((terme(c)=='t')&((c==1)|(terme(c-1)~='/')))
                      % Recherche de la longueur du nombre du "t" à
effacer
                      d=1;
                      while
(((c+d)<=lterme)&(length(str2num(terme(c+d)))))
                         d=d+1;

                      end
                      % d pointe le caractère après ancienne constante
                      % Effaçage du "l"
                      %(si le caractère d'avant l'ancienne constante est
un *, alors l'enlever)
                      if ((c==1)|(terme(c-1)=='*'))
                         if (c~=1) % test c==1 pour éviter erreur si c=1
pour terme(c-1)...
                            enlevemul=-1;
                         end
                      else
                         enlevemul=0;
                      end
                      if (c==1)
                         terme=terme(c+d:lterme);
                      elseif (c+d>=lterme)
                         terme=terme(1:(c-1)+enlevemul);
                      else
                         terme=[terme(1:(c-
1)+enlevemul),terme(c+d:lterme)];
                      end
                      lterme=length(terme); % Longueur du nouveau terme
                      c=c-1; % Si on enlève un terme, un regarde encore à
la même place pour savoir s'il y a une autre constante

                   end

                   c=c+1;
                end

                % Vérification (y a-t-il un * juste après le (+ ou -)
                if (lterme==0) % pour éviter les erreurs lorsque lterme=0
                elseif (terme(1)=='*')
                   terme=terme(2:lterme);
                   lterme=length(terme);
                end

                % Écriture du nouveau terme
                if (lterme~=0)
                   terme=[terme,'*',ncons];
                   lterme=length(terme);
                else
                   terme=ncons;
                   lterme=length(terme);
                end
```

```
                % Modification de svar (remplacement de l'ancien terme
par le nouveau)
                if (pos==1) % si premier terme
                   svar=[terme,svar(fin+1:long)];
                elseif ((pos+ltermea-1)==long) % si dernier terme
                   svar=[svar(1:debut-1),terme];
                else
                   svar=[svar(1:debut-1),terme,svar(fin+1:long)]; % si
terme intermédiaire
                end

                % Nouvelle longueur de svar
                long=length(svar);

            end

         end


         pos=pos+1;

      end

      % Transformation de l'élément de la variable modifié en symbolique
      vsor(a,b)=sym(svar);

   end
end
```

Coquille de la routine de calcul de la dynamique inverse servant à calculer la matrice des masses

```
/*
 * DYNDIRLE2_M : Dynamique directe du doigt selon la méthode de
Lagrange-Euler :
 *               Sous-fonction de calcul de la matrice des masses
 *
 *   Entrées : position articulaire
 *             masses de l'articulation
 *             vecteur de calcul des sinus et cosinus
 *
 *
 *   Sortie : 1-M^2 -> matrice des masses/inerties
 *
 *   Par : Martin de Montigny
 *   UQTR : Laboratoire de commande
 *   Copyright (c) février 1999
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME dyndirle_m
```

```
#include "simstruc.h"
#include "math.h"
/*définition des constantes*/
/*1*/

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
/*nombre d'entrées et de sorties*/
/*2*/
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
/*équations*/
/*3*/
```

```
}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Coquille de la routine de calcul de la dynamique inverse servant à calculer le vecteur des termes de gravité, de Coriolis et de forces centrifuges

```
/*
 * DYNDIRLE_VG : Dynamique directe du doigt selon la méthode de
Lagrange-Euler :
 *               Sous-fonction de calcul du vecteur de termes de vitesse
et de
 *               coriolis et de gravité.
 *
 *   Entrées : q (position articulaire)
 *             qp (vitesse articulaire)
 *             masses des membres
 *             vecteur des termes trigonométriques
 *
```

```
 *   Sorties : 1-M -> Couple représentant les termes de vitesse/coriolis
et de gravité
 *
 *   Par : Martin de Montigny
 *   UQTR : Laboratoire de commande
 *   Copyright (c) février 1999
 *   Tous droits réservés
 *   version 1.1
 */

#define S_FUNCTION_NAME dyndirle_vg

#include "simstruc.h"
#include "math.h"
/* Définition des constantes */
/*1*/

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
/* nombre d'entrées et de sorties */
/*2*/
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
```

```
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
/* Algorithme */
/*3*/

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Coquille de la routine de calcul des vitesse et accélération articulaires en fonction des vitesse et accélération galliléennes et de la position articulaire

```
/*
 * Jacobien_vitesse_acceleration : Calcul de la vitesse et de
l'accélération articulaire en fonction
```

```
 *                                  de la vitesse et de l'accélération
cartésienne
 *
 *   Entrées : Vitesse cartésienne
 *             Accélération cartésienne
 *             Position articulaire
 *
 *   Sortie  : 1 à 2M    -> Vitesses et accélérations articulaires
 *
 *
 *   Par : Martin de Montigny
 *   UQTR : Laboratoire de commande
 *   Copyright (c) 1998-99
 *   Tous droits réservés
 *   version 2.0
 */

#define S_FUNCTION_NAME jacobien_vitacc
/*1*/

#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
/*2*/
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
```

```
/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

/*3*/


}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Coquille de la routine de calcul de la loi de commande utilisant l'algorithme de Slotine et Li

```
/*
```

```
 * LOI_COMMANDE : Calcul de la loi de commande (Tcom)
 *
 *   Entrées : a (paramètres estimés : masses)
 *             qrp (vitesse articulaire modifiée)
 *             qrpp-sd (accélération articulaire modifiée et comprenant
un contrôleur PD)
 *             qp (vitesse articulaire)
 *             vecteur de précalcul de sinus et de consinus de q
(position articulaire)
 *
 *   Sorties : 1-3 -> Couple de commande
 *
 *
 *   Par : Martin de Montigny
 *   UQTR : Laboratoire de commande
 *   Copyright (c) 1998-99
 *   Tous droits réservés
 *   version 2.0
 */

#define S_FUNCTION_NAME Loi_commande
/*1*/


#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
/*2*/
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
```

```
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

/*3*/

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Coquille de la routine de calcul de calcul de l'opposé de la matrice inverse de covariance (-P) pour la loi d'adaptation de Slotine et Li

```
/*
 * CALCUL_Pp : Calcul de P dérivé (pour calculer la derivée de a)
 *
 *   Entrées : matrice W
 *             matrice P
 *             rho (facteur d'oubli)
 *             k0 (limite de module de la matrice P)
 *
 *   Sortie : matrice P dérivée
 *
 *
 *   Par : Martin de Montigny
 *   UQTR : Laboratoire de commande
 *   Copyright (c) 1998-99
 *   Tous droits réservés
 *   version 2.0
 */

#define S_FUNCTION_NAME calcul_pp

#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
/*1*/
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
```

```
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

/*2*/

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Coquille de la routine de calcul de la matrice des signaux W pour l'algorithme d'adaptation des paramètres de Slotine et Li

```
/*
 * CALCUL_W : Calcul de w (pour calculer T filtré)
 *
 *   Entrées : qp (vitesse articulaire)
 *             g (constante gravitationnelle)
 *             vecteur de précalcul de sinus et de cosinus de q
(position articulaire)
 *             lambda (fréquence du filtre)
 *
 *   Sorties : 1-9 -> Partie 1 de la matrice W (à filtrer)
 *             10-18 -> Partie 2 de la matrice W
 *
 *
 *   Par : Martin de Montigny
 *   UQTR : Laboratoire de commande
 *   Opal-rt
 *   Copyright (c) 1998-99
 *   Tous droits réservés
 *   version 2.0
 */

#define S_FUNCTION_NAME Calcul_w
/*1*/


#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
/*2*/
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
```

```
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

/*3*/

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
```

```
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Coquille de la routine de calcul de la dérivée des paramètres identifiés pour l'algorithme d'identification des paramètres de Slotine et Li

```
/*
 * calcul_y_a : Calcul de Y et a (partiel) (pour calculer la derivée de
a)
 *
 *   Entrées : qp (vitesse articulaire)
 *             e (erreur de couple causée par l'erreur d'observation des
paramètres)
 *             s (erreur de poursuite)
 *             qrp (vitesse articulaire modifiée)
 *             qrpp (accélération articulaire modifiée)
 *             matrice W
 *             matrice P
 *             matrice R (constante)
 *             vecteur de précalcul de sinus et de cosinus de q
(position articulaire)
 *
 *   Sortie : 1-3 -> a (paramètres estimés : masses)
 *
 *   Par : Martin de Montigny
 *   UQTR : Laboratoire de commande
 *   Copyright (c) 1998-99
 *   Tous droits réservés
 *   version 2.0
 */

#define S_FUNCTION_NAME calcul_y_a
/*1*/


#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
/*2*/
```

```
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

/*3*/

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
```

```
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Coquille de la routine de calcul d'un produit matrice - vecteur de dimension quelconque (la dimension est fixée à la compilation)

```
/*
 * prodmatvectn : Calcul du pruduit matrice vecteur d'ordre n
 *
 *   Entrées : Matrice d'entrée (ligne par ligne)
 *             Vecteur d'entrée
 *
 *   Sortie : Vecteur de sortie
 *
 *   Par : Martin de Montigny
 *   UQTR : Laboratoire de commande
 *   Copyright (c) Avril 1999
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME prodmatvectn

#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
/*1*/
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
```

```
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

/*2*/

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}
```

```
/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Programme Perl de modification du fichier de configuration de « divmat.c » (largeur des entrées/ sorties)

```
#!c:\matlab\bin\perl

my($nout2, $nout, $nin);

$nout = $ARGV[0];
if(! $nout)
{
    print "ERREUR \n";
}
$nout2 = $nout*$nout;
$nin = $nout2+$nout;

open(OUT, ">libconst.h") || die "Cannot create libconst.h";
print OUT "\#define  NIN     $nin\n\#define NOUT    $nout\n\#define
NOUT2    $nout2\n";

close(OUT);
```

Programme Perl de modification du programme Matlab « cin.m » de lancement du calcul de la cinématique directe

```
#c:\matlab\bin\perl

# Programme de création du fichier cinx.m pour la cinématique
# directe.  Les entrées sont :
#
```

```
# 1) type de manipulateur ex: rrr
#
# Par : Martin de Montigny
# Janvier 1999
# Université du Québec à Trois-Rivières

my ($type, $tabdefbra);
$nom=$ARGV[0];

if (! $nom)
{
   print "Erreur (entrées incorrectes) \n";
}

open (INS,"<defbra.m") or die "fichier source defbra non accessible";
open(INEQ1,"<temp.txt") or die "fichier source des équations de defbra
non accessible";
open(OUT,">$nom") or die "impossible de créer ou d'écraser le fichier
befbrax";

while(<INS>)
{
   if (/%\*%\*%1/)
   {
      print OUT $_;
      while(<INEQ1>)
      {
         print OUT $_;
      }
   }
   else
   {
      print OUT $_;
   }
}

close(OUT);
close(INEQ1);
close(INS);
```

Programme Perl de modification du programme Matlab « defbra.m » de configuration du manipulateur pour le calcul de la cinématique directe

```
#c:\matlab\bin\perl

# Programme de création du fichier defmanx.m de configuration du
manipulateur pour la cinématique
# directe.  Les entrées sont :
#
# 1) nom du fichier .m à créer (sans l'extension)
#
# Par : Martin de Montigny
```

```
# Janvier 1999
# Université du Québec à Trois-Rivières

my ($nom, $nomcin, $nomsave);
$nom=$ARGV[0];
$nomcin=$ARGV[1];
$nomsave=$ARGV[2];

if ((! $nom)||(! $nomcin)||(! $nomsave))
{
   print "Erreur (entrées incorrectes) \n";
}

open (INS,"<defbra.m") or die "fichier source defbra non accessible";
open(INEQ1,"<temp.txt") or die "fichier source des équations de defbra
non accessible";
open(OUT,">$nom.m") or die "impossible de créer ou d'écraser le fichier
befbrax";

while(<INS>)
{
   if (/%\*%\*%1/)
   {
      print OUT $_;
      while(<INEQ1>)
      {
         print OUT $_;
      }
   }
   else
   {
      print OUT $_;
   }
}

close(OUT);
close(INEQ1);
close(INS);

open (INS,"<cin.m") or die "fichier source cin non accessible";
open (OUT,">$nomcin.m") or die "impossible de créer ou d'écraser le
fichier cinx";

while(<INS>)
{
   if (/%\*%\*%1/)
   {
      print OUT $_;
      print OUT "[M,N,Alphaim1,Aim1,Di,Thetai,Ki]=$nom;\n";
   }
   elsif (/%\*%\*%2/)
   {
      print OUT $_;
      print OUT "save $nomsave\n";
   }
   else
   {
```

```
      print OUT $_;
   }
}

close(OUT);
close(INS);
```

Programme Perl de modification du programme Matlab « defman.m » de configuration du manipulateur pour le calcul de la dynamique inverse

```
#c:\matlab\bin\perl

# Programme de création du fichier defmanx.m de configuration du
manipulateur pour la dynamique
# inverse.  Les entrées sont :
#
# 1) nom du fichier .m à créer (sans l'extension)
# 2) nom du fichier de sauvegarde des données de la cinématique
# 3) nombre d'articulations actives ("M" selon la norme du menu GUI :
Nombre d'articulations actives)
#
# Par : Martin de Montigny
# Février 1999
# Université du Québec à Trois-Rivières

my ($nom, $nomsave, $M);
$nom=$ARGV[0];
$nomsave=$ARGV[1];
$M=$ARGV[2];

if ((! $nom)||(! $nomsave)||(! $M))
{
   print "Erreur (entrées incorrectes) \n";
}

open (INS,"<defman.m") or die "fichier source defman non accessible";
open(INEQ1,"<temp1.txt") or die "fichier source 1 de defman non
accessible";
open(INEQ2,"<temp2.txt") or die "fichier source 2 de defman non
accessible";
open(INEQ3,"<temp3.txt") or die "fichier source 3 de defman non
accessible";
open(OUT,">$nom.m") or die "impossible de créer ou d'écraser le fichier
defmanx";

while(<INS>)
{
   if (/%\*%\*%3/)
   {
      print OUT $_;
      while(<INEQ1>)
```

```
        {
            print OUT $_;
        }
    }
    elsif (/%\*%\*%4/)
    {
        print OUT $_;
        while(<INEQ2>)
        {
            print OUT $_;
        }
    }
    elsif (/%\*%\*%5/)
    {
        print OUT $_;
        while(<INEQ3>)
        {
            print OUT $_;
        }
    }
    elsif (/%\*%\*%1/)
    {
        print OUT $_;
        print OUT "load $nomsave;\n";
    }
    elsif (/%\*%\*%2/)
    {
        print OUT $_;
        print OUT "Mi = [1";
        for ($a=2;$a<=$M;$a++)
        {
            print OUT ";1";
        }
        print OUT "];\n";
    }
    else
    {
        print OUT $_;
    }
}

close(OUT);
close(INEQ1);
close(INEQ2);
close(INEQ3);
close(INS);
```

Programme principal de génération de code sous la formulation de Newton-Euler

```
% genalgo_ne.m
%
% Par : Martin de Montigny
% Date : aout 1998
%
% Générateur automatique d'algorithme de Newton-Euler optimisé en
fonction des
% paramètres DH-modifiés d'un manipulateur à N articulations en série.
%
% Les paramètres ajustables (à l'intérieur du programme) sont :
%
% nomd : nom du fichier destination (créé ou écrasé) des équations de
dynamique inverse
% nomc : nom du fichier destination (créé ou écrasé) des équations de
cinématique directe
% noms1 : nom du fichier source sans équation de dynamique inverse,
itération de base à effecteur
% noms2 : nom du fichier source sans équation de dynamique inverse,
itération de l'effecteur à la base
% noms3 : nom du fichier source sans équation de cinématique directe
% nomlibne : nom des librairies de dynamique inverse (M+1 librairies)
% nomlibcin : nom de la librairie de cinématique directe

% Nom des fichiers à lire et écrire (base sans extension)
nomd='doigtne'; % fichier code C destination (équations dynamiques)
nomc='doigtcin'; % fichier destination (équations cinématiques)
noms1='dyninvbae'; % fichier source C itération base à effecteur
noms2='dyninveab'; % fichier source C itération effecteur à base
noms3='cin_dir'; % fichier source C de cinématique directe
nomlibne='libne'; % nom de la librairie à produire (dynamique)
nomlibcin='libcin'; % nom de la librairie à produire (cinématique)

% Initialisation des variables symboliques
syms Rivim1 r11 r12 r21 r22 r23 r31 r32 r33 real
syms r011 r012 r013 r021 r022 r023 r031 r032 r033 real
syms wim1 wim11 wim12 wim10 qid real
syms wpim1 wpim10 wpim11 wpim12 qidd real
syms Pii pi0 pi1 pi2 real
syms vpim1 vpim10 vpim11 vpim12 real
syms Pci pci0 pci1 pci2 real
syms Ten ten0 ten1 ten2 ten3 ten4 ten5 ten6 ten7 ten8 real

syms wic wic0 wic1 wic2 real
syms wpic wpic0 wpic1 wpic2 real
syms vpic vpic0 vpic1 vpic2 real
syms vpcic vpcic0 vpcic1 vpcic2 real
syms masse real

syms Rip1vi fip1 nip1 Pip1 real
syms fip10 fip11 fip12 nip10 nip11 nip12 pip10 pip11 pip12 real
```

```
syms Fic Fic0 Fic1 Fic2 Nic Nic0 Nic1 Nic2 real

syms pim0 pim1 pim2 real

syms cthetai sthetai calphaim1 salphaim1 dii aim1t diii real
syms dit qit cthetaii sthetaii sthetaipqi cthetaipqi real


% Message à l'utilisateur
disp('Calcul des équations de Newton-Euler');

% Boucle de création pour les M articulations (articulations actives)
for a=1:M

   % Ouverture du fichier de l'articulation M
   eval(['h=fopen(''',nomd,num2str(a),'.txt'',''w'');']);

   % Écriture de l'entête du fichier
   eval(['fprintf(h,''Fichier de calculs de l''''articulation ',...
      num2str(a),'\n\n'');']);

   % Création des équations de vitesse widi de l'articulation M

   % Vérification du type d'articulation
   if (ki(a)==1)
      tid0=2;
      did0=0;
   else
      tid0=0;
      did0=1;
   end

   Rivim1n=abs(sign([(tid0+cos(thetai(a))),(tid0+-sin(thetai(a))),0;...

(tid0+sin(thetai(a)))*cos(alphaim1(a)),(tid0+cos(thetai(a)))*cos(alphaim
1(a)),...
      -sin(alphaim1(a));...

      (tid0+sin(thetai(a)))*sin(alphaim1(a)),(tid0+cos(thetai(a)))*sin(a
lphaim1(a)),...
      cos(alphaim1(a))]));

   Rivim1=[r11,r12,0;r21,r22,r23;r31,r32,r33].*Rivim1n;

   wim1=[wim10;wim11;wim12];

   wi=Rivim1.'*wim1+[0;0;qid]*ki(a); %calcul de la vitesse rotative
   wi=simple(wi);

   fprintf(h,['widi=\n']);
   for b=1:3
      wit=[char(wi(b)),';\n'];
      wit=strrep(wit,'r11','u[24]');
      wit=strrep(wit,'r12','u[25]'); %r13=0 toujours...
      wit=strrep(wit,'r21','u[27]');
      wit=strrep(wit,'r22','u[28]');
      wit=strrep(wit,'r23','u[29]');
```

```
   wit=strrep(wit,'r31','u[30]');
   wit=strrep(wit,'r32','u[31]');
   wit=strrep(wit,'r33','u[32]');

   wit=strrep(wit,'wim10','u[0]');
   wit=strrep(wit,'wim11','u[1]');
   wit=strrep(wit,'wim12','u[2]');

   wit=strrep(wit,'qid','u[9]');

   fprintf(h,['   y[',num2str(b-1),']=']);
   fprintf(h,wit);
end

fprintf(h,'\n');

% Création des équations d'accélération rotative wpidi

wpim1=[wpim10;wpim11;wpim12];

wpi=Rivim1.'*wpim1+ki(a)*(cross(Rivim1.'*wim1,[0;0;qid])+[0;0;qidd]);
wpi=simple(wpi);

fprintf(h,['wpidi=\n']);
for b=1:3
   wpit=[char(wpi(b)),';\n'];
   wpit=strrep(wpit,'r11','u[24]');
   wpit=strrep(wpit,'r12','u[25]');
   wpit=strrep(wpit,'r21','u[27]');
   wpit=strrep(wpit,'r22','u[28]');
   wpit=strrep(wpit,'r23','u[29]');
   wpit=strrep(wpit,'r31','u[30]');
   wpit=strrep(wpit,'r32','u[31]');
   wpit=strrep(wpit,'r33','u[32]');

   wpit=strrep(wpit,'wim10','u[0]');
   wpit=strrep(wpit,'wim11','u[1]');
   wpit=strrep(wpit,'wim12','u[2]');

   wpit=strrep(wpit,'wpim10','u[3]');
   wpit=strrep(wpit,'wpim11','u[4]');
   wpit=strrep(wpit,'wpim12','u[5]');

   wpit=strrep(wpit,'qidd','u[10]');
   wpit=strrep(wpit,'qid','u[9]');

   fprintf(h,['   y[',num2str(b+2),']=']);
   fprintf(h,wpit);
end

fprintf(h,'\n');


% Création des équations d'accélération linéaire vpidi

% Test pour savoir si wic contient des valeurs nulles
wic=[wic0;wic1;wic2];
```

```
   wic=wic.*not(wi==0);

   Piin=([aim1(a);-
sin(alphaim1(a))*di(a);cos(alphaim1(a))*di(a)]~=0)+...
      [0;-sin(alphaim1(a))*did0;cos(alphaim1(a))*did0])~=0;
   Pii=Piin.*[pi0;pi1;pi2];

   vpim1=[vpim10;vpim11;vpim12];

   vpi=Rivim1.'*(cross(wpim1,Pii)+cross(wim1, ...
      cross(wim1,Pii))+vpim1)+...
      (1-ki(a))*(2*cross(wic,[0;0;qid])+[0;0;qidd]);
   vpi=simple(vpi);
   vpi=transpm(vpi); %enlève les puissances et les remplaces par des
multiplications...

   fprintf(h,['vpidi=\n']);
   for b=1:3
      vpit=[char(vpi(b)),';\n'];
      vpit=strrep(vpit,'r11','u[24]');
      vpit=strrep(vpit,'r12','u[25]');
      vpit=strrep(vpit,'r21','u[27]');
      vpit=strrep(vpit,'r22','u[28]');
      vpit=strrep(vpit,'r23','u[29]');
      vpit=strrep(vpit,'r31','u[30]');
      vpit=strrep(vpit,'r32','u[31]');
      vpit=strrep(vpit,'r33','u[32]');

      vpit=strrep(vpit,'wim10','u[0]');
      vpit=strrep(vpit,'wim11','u[1]');
      vpit=strrep(vpit,'wim12','u[2]');

      vpit=strrep(vpit,'wpim10','u[3]');
      vpit=strrep(vpit,'wpim11','u[4]');
      vpit=strrep(vpit,'wpim12','u[5]');

      vpit=strrep(vpit,'vpim10','u[6]');
      vpit=strrep(vpit,'vpim11','u[7]');
      vpit=strrep(vpit,'vpim12','u[8]');

      vpit=strrep(vpit,'pi0','u[42]');
      vpit=strrep(vpit,'pi1','u[43]');
      vpit=strrep(vpit,'pi2','u[44]');

      vpit=strrep(vpit,'wic0','y[0]');
      vpit=strrep(vpit,'wic1','y[1]');
      vpit=strrep(vpit,'wic2','y[2]');

      vpit=strrep(vpit,'qidd','u[10]');
      vpit=strrep(vpit,'qid','u[9]');

      fprintf(h,['   y[',num2str(b+5),']=']);
      fprintf(h,vpit);
   end

   fprintf(h,'\n');
```

```
% Calcul de l'accélération linéaire de la masse du membre

Pci=[pci0;pci1;pci2];
% Test pour voir si wpidi et vpidi contiennent des valeurs nulles...
wpic=[wpic0;wpic1;wpic2];
vpic=[vpic0;vpic1;vpic2];

wpic=wpic.*not(wpi==0);
vpic=vpic.*not(vpi==0);

vpci=cross(wpic,Pci)+cross(wic,cross(wic,Pci))+vpic;
vpci=simple(vpci);

% Calcul des forces et couples d'inertie

vpcic=[vpcic0;vpcic1;vpcic2];
% Test pour voir si vpcidi contient des valeurs nulles...
vpcic=vpcic.*(vpci~=0);

Tenn=([tenseur((a-1)*9+1:(a-1)*9+3)';tenseur((a-1)*9+4:(a-1)*9+6)';
...
      tenseur((a-1)*9+7:a*9)']~=0);
Ten=Tenn.*[ten0,ten1,ten2;ten3,ten4,ten5;ten6,ten7,ten8];

Fi=masse*vpci;
% Enlève les exposants (non supportés par le compilateur mex)
Fi=transpm(Fi);

Ni=Ten*wpic+cross(wic,Ten*wic);
Ni=simple(Ni);

fprintf(h,['Fi=\n']);
for b=1:3
   Fit=[char(Fi(b)),';\n'];

   Fit=strrep(Fit,'wic0','y[0]');
   Fit=strrep(Fit,'wic1','y[1]');
   Fit=strrep(Fit,'wic2','y[2]');

   Fit=strrep(Fit,'wpic0','y[3]');
   Fit=strrep(Fit,'wpic1','y[4]');
   Fit=strrep(Fit,'wpic2','y[5]');

   Fit=strrep(Fit,'vpic0','y[6]');
   Fit=strrep(Fit,'vpic1','y[7]');
   Fit=strrep(Fit,'vpic2','y[8]');

   Fit=strrep(Fit,'pci0','u[12]');
   Fit=strrep(Fit,'pci1','u[13]');
   Fit=strrep(Fit,'pci2','u[14]');

   Fit=strrep(Fit,'masse','u[11]');

   fprintf(h,['   y[',num2str(b+8),']=']);
   fprintf(h,Fit);
end
```

```
   fprintf(h,'\n');

   fprintf(h,['Ni=\n']);
   for b=1:3
      Nit=[char(Ni(b)),';\n'];

      Nit=strrep(Nit,'wic0','y[0]');
      Nit=strrep(Nit,'wic1','y[1]');
      Nit=strrep(Nit,'wic2','y[2]');

      Nit=strrep(Nit,'wpic0','y[3]');
      Nit=strrep(Nit,'wpic0','y[4]');
      Nit=strrep(Nit,'wpic0','y[5]');

      Nit=strrep(Nit,'ten0','u[15]');
      Nit=strrep(Nit,'ten1','u[16]');
      Nit=strrep(Nit,'ten2','u[17]');
      Nit=strrep(Nit,'ten3','u[18]');
      Nit=strrep(Nit,'ten4','u[19]');
      Nit=strrep(Nit,'ten5','u[20]');
      Nit=strrep(Nit,'ten6','u[21]');
      Nit=strrep(Nit,'ten7','u[22]');
      Nit=strrep(Nit,'ten8','u[23]');

      fprintf(h,['   y[',num2str(b+11),']=']);
      fprintf(h,Nit);
   end

   fprintf(h,'\n');

   % Calcul des forces et couples externes

   aa=a+1;

   % Vérification du type d'articulation
   if (ki(aa)==1)
      tid0=2;
      did0=0;
   else
      tid0=0;
      did0=1;
   end

   Rip1vin=abs(sign([(tid0+cos(thetai(aa))),(tid0+-
sin(thetai(aa))),0;...

(tid0+sin(thetai(aa)))*cos(alphaim1(aa)),(tid0+cos(thetai(aa)))*cos(alph
aim1(aa)),...
      -sin(alphaim1(aa));...

      (tid0+sin(thetai(aa)))*sin(alphaim1(aa)),(tid0+cos(thetai(aa)))*si
n(alphaim1(aa)),...
      cos(alphaim1(aa))]));

   Rip1vi=[r11,r12,0;r21,r22,r23;r31,r32,r33].*Rip1vin;
```

```
   Pip1n=([aim1(aa);-
sin(alphaim1(aa))*di(aa);cos(alphaim1(aa))*di(aa)]+...
      [0;-sin(alphaim1(aa))*did0;cos(alphaim1(aa))*did0])~=0;
   Pip1=Pip1n.*[pip10;pip11;pip12];

   fip1=[fip10;fip11;fip12];
   nip1=[nip10;nip11;nip12];

   Fic=[Fic0;Fic1;Fic2];
   Nic=[Nic0;Nic1;Nic2];

   Fic=Fic.*([Fi(1);Fi(2);Fi(3)]~=0); %prendre le nom des variables déja
calculées
                                      %et simplifier les éléments nuls
   Nic=Nic.*([Ni(1);Ni(2);Ni(3)]~=0);

   fi=simple(Rip1vi*fip1+Fic);
   ni=Nic+Rip1vi*nip1+cross(Pci,Fic)+cross(Pip1,(Rip1vi*fip1));

   fprintf(h,['fi=\n']);
   for b=1:3
      fit=[char(fi(b)),';\n'];

      fit=strrep(fit,'r11','u[15]');
      fit=strrep(fit,'r12','u[16]');
      fit=strrep(fit,'r21','u[18]');
      fit=strrep(fit,'r22','u[19]');
      fit=strrep(fit,'r23','u[20]');
      fit=strrep(fit,'r31','u[21]');
      fit=strrep(fit,'r32','u[22]');
      fit=strrep(fit,'r33','u[23]');

      fit=strrep(fit,'fip10','u[0]');
      fit=strrep(fit,'fip11','u[1]');
      fit=strrep(fit,'fip12','u[2]');

      fit=strrep(fit,'Fic0','u[6]');
      fit=strrep(fit,'Fic1','u[7]');
      fit=strrep(fit,'Fic2','u[8]');

      fprintf(h,['   y[',num2str(b),']=']);
      fprintf(h,fit);
   end

   fprintf(h,'\n');

   fprintf(h,['ni=\n']);
   for b=1:3
      nit=[char(ni(b)),';\n'];

      nit=strrep(nit,'r11','u[15]');
      nit=strrep(nit,'r12','u[16]');
      nit=strrep(nit,'r21','u[18]');
      nit=strrep(nit,'r22','u[19]');
      nit=strrep(nit,'r23','u[20]');
      nit=strrep(nit,'r31','u[21]');
      nit=strrep(nit,'r32','u[22]');
```

```
      nit=strrep(nit,'r33','u[23]');

      nit=strrep(nit,'Nic0','u[9]');
      nit=strrep(nit,'Nic1','u[10]');
      nit=strrep(nit,'Nic2','u[11]');

      nit=strrep(nit,'fip10','u[0]');
      nit=strrep(nit,'fip11','u[1]');
      nit=strrep(nit,'fip12','u[2]');

      nit=strrep(nit,'Fic0','u[6]');
      nit=strrep(nit,'Fic1','u[7]');
      nit=strrep(nit,'Fic2','u[8]');

      nit=strrep(nit,'nip10','u[3]');
      nit=strrep(nit,'nip11','u[4]');
      nit=strrep(nit,'nip12','u[5]');

      nit=strrep(nit,'pip10','u[24]');
      nit=strrep(nit,'pip11','u[25]');
      nit=strrep(nit,'pip12','u[26]');

      nit=strrep(nit,'pci0','u[12]');
      nit=strrep(nit,'pci1','u[13]');
      nit=strrep(nit,'pci2','u[14]');

      fprintf(h,['   y[',num2str(b+3),']=']);
      fprintf(h,nit);
   end

   % Calcul du couple articulaire
   fprintf(h,'\n');
   fprintf(h,['Ti=\n']);
   if (ki(a)==1)
      fprintf(h,['y[6]',';']);
   else
      fprintf(h,['y[3]',';']);
   end

   fclose(h);

end


% Création des M fichiers source d'algorithmes d'itérations base à
effecteur
% et effecteur à base | Compilation des fichiers source

disp('Création et compilation des fichiers C-MEX');
for a=1:M

   vrainom1=[noms1,num2str(a),'.c'];
   vrainom2=[noms2,num2str(a),'.c'];
   nomeq=[nomd,num2str(a),'.txt'];
   eval(['!perl modsource.pl ',vrainom1,' ',vrainom2,' ',nomeq]); %
Création
   eval(['mex ',vrainom1]);eval(['mex ',vrainom2]); %Compilation
```

```
end

% Production des librairies contenant les blocs d'appel des
% S-Function de dynamique inverse

eval(['!perl prodlibne.pl ',num2str(M),' ',nomlibne,'.mdl']);



% Génération de l'algorithme de cinématique directe

% Message à l'utilisateur
disp('Calcul des équations de cinématique directe');

Rim1v0n=0;
Pim1d0n=0;
for a=1:N

   % Ouverture du fichier de l'articulation a
   eval(['h=fopen(''',nomc,num2str(a),'.txt'',''w'');']);

   % Écriture de l'entête du fichier
   eval(['fprintf(h,''Fichier de calculs cinématique de
l''''articulation ',...
      num2str(a),'\n\n'');']);

   % Calcul trigonométriques initiaux
   if ki(a)==0
      diii=dit+qit; % di=qi+di = var.articulaire + offset
      syms dii real % variable di utilisée plus loin
      if thetai(a)==0
         sthetaii(1)=0; %utilisé localement
                  sthetai(1)=0; %utilisé ailleur (autres calculs, même
fonction)
         cthetaii(1)=1;
                  cthetai(1)=1;
      else
         syms sthetai cthetai sthetaii cthetaii real
      end

   else
      sthetaii=sthetaipqi;
      cthetaii=cthetaipqi;
            syms sthetai cthetai real

      if di(a)==0
         diii(1)=0;
                  dii(1)=0;
      else
         diii=dit;
                  syms dii real
      end
   end

   if alphaim1(a)==0
      salphaim1(1)=0;
```

```
   calphaim1(1)=1;
else
   syms salphaim1 calphaim1 real
end

% Écriture de "trigo:"
fprintf(h,'trigo:\n');

% Écriture de di
ditt=[char(diii),';\n'];
ditt=strrep(ditt,'dit','u[16]');
ditt=strrep(ditt,'qit','u[13]');
ditt=strrep(ditt,'=0;','=0.0;');
fprintf(h,['   di=']);
fprintf(h,ditt);

sthetait=[char(sthetaii),';\n'];
sthetait=strrep(sthetait,'sthetaipqi','sin(u[17]+u[13])');
sthetait=strrep(sthetait,'sthetaii','sin(u[17])');
sthetait=strrep(sthetait,'=0;','=0.0;');
sthetait=strrep(sthetait,'=1;','=1.0;');
fprintf(h,['   sthetai=']);
fprintf(h,sthetait);

% Écriture de cthetai
cthetait=[char(cthetaii),';\n'];
cthetait=strrep(cthetait,'cthetaipqi','cos(u[17]+u[13])');
cthetait=strrep(cthetait,'cthetaii','cos(u[17])');
cthetait=strrep(cthetait,'=0;','=0.0;');
cthetait=strrep(cthetait,'=1;','=1.0;');
fprintf(h,['   cthetai=']);
fprintf(h,cthetait);

% Écriture de salphaim1
salphaim1t=[char(salphaim1),';\n'];
salphaim1t=strrep(salphaim1t,'salphaim1','sin(u[15])');
salphaim1t=strrep(salphaim1t,'=0;','=0.0;');
salphaim1t=strrep(salphaim1t,'=1;','=1.0;');
fprintf(h,['   salphaim1=']);
fprintf(h,salphaim1t);

% Écriture de calphaim1
calphaim1t=[char(calphaim1),';\n'];
calphaim1t=strrep(calphaim1t,'calphaim1','cos(u[15])');
calphaim1t=strrep(calphaim1t,'=0;','=0.0;');
calphaim1t=strrep(calphaim1t,'=1;','=1.0;');
fprintf(h,['   calphaim1=']);
fprintf(h,calphaim1t);


% Écriture de "Rivim1="
fprintf(h,'\nRivim1=\n');

% Écriture de l'expression de Riv0

if (ki(a)==1)
   tid0=2;
```

```
      did0=0;
   else
      tid0=0;
      did0=1;
   end

   Rivim1n=abs(sign([(tid0+cos(thetai(a))),(tid0-sin(thetai(a))),0;...

(tid0+sin(thetai(a)))*cos(alphaim1(a)),(tid0+cos(thetai(a)))*cos(alphaim
1(a)),...
      -sin(alphaim1(a));...

      (tid0+sin(thetai(a)))*sin(alphaim1(a)),(tid0+cos(thetai(a)))*sin(a
lphaim1(a)),...
      cos(alphaim1(a))]));

   Rivim1=[cthetai,-sthetai,0;sthetai*calphaim1,cthetai*calphaim1, ...
      -salphaim1;sthetai*salphaim1,cthetai*salphaim1, ...
      calphaim1].*Rivim1n;

   if a==1
      Rim1v0=[rotbase(1:3)';rotbase(4:6)';rotbase(7:9)'];
   else
      Rim1v0=(Riv0~=0).*[r011,r012,r013;r021,r022,r023;r031,r032,r033];
   end

   Rim1v0n=(Rim1v0~=0); % Mise à 0 des éléments nuls de Riv0

Riv0=(([r011,r012,r013;r021,r022,r023;r031,r032,r033].*Rim1v0n)*Rivim1);
   Riv0t2=[Riv0(1,1:3)';Riv0(2,1:3)';Riv0(3,1:3)'];

   Rivim1t2=[Rivim1(1,1:3)';Rivim1(2,1:3)';Rivim1(3,1:3)'];
   for b=1:9
      Rivim1t=[char(Rivim1t2(b,1)),';\n'];

      Rivim1t=strrep(Rivim1t,'qit','u[13]');
      Rivim1t=strrep(Rivim1t,'=0;','=0.0;');
      Rivim1t=strrep(Rivim1t,'=1;','=1.0;');

      fprintf(h,['   y[',num2str(b+12),']=']);
      fprintf(h,Rivim1t);
   end


   % Écriture de "Riv0="
   fprintf(h,'\nRiv0=\n');

   for b=1:9
      Riv0t=[char(Riv0t2(b,1)),';\n'];

      Riv0t=strrep(Riv0t,'r11','y[13]');
      Riv0t=strrep(Riv0t,'r12','y[14]');
      Riv0t=strrep(Riv0t,'r21','y[16]');
      Riv0t=strrep(Riv0t,'r22','y[17]');
      Riv0t=strrep(Riv0t,'r23','y[18]');
      Riv0t=strrep(Riv0t,'r31','y[19]');
      Riv0t=strrep(Riv0t,'r32','y[20]');
```

```
      Riv0t=strrep(Riv0t,'r33','y[21]');

      Riv0t=strrep(Riv0t,'r011','u[1]');
      Riv0t=strrep(Riv0t,'r012','u[2]');
      Riv0t=strrep(Riv0t,'r013','u[3]');
      Riv0t=strrep(Riv0t,'r021','u[4]');
      Riv0t=strrep(Riv0t,'r022','u[5]');
      Riv0t=strrep(Riv0t,'r023','u[6]');
      Riv0t=strrep(Riv0t,'r031','u[7]');
      Riv0t=strrep(Riv0t,'r032','u[8]');
      Riv0t=strrep(Riv0t,'r033','u[9]');

      Riv0t=strrep(Riv0t,'=0;','=0.0;');

      fprintf(h,['   y[',num2str(b),']=']);
      fprintf(h,Riv0t);
   end

   Pidim1n=([aim1(a);-
sin(alphaim1(a))*di(a);cos(alphaim1(a))*di(a)]~=0)|...
      [0;did0*sin(alphaim1(a));did0*cos(alphaim1(a))];

   Pidim1=[aim1t;-salphaim1*dii;calphaim1*dii].*Pidim1n;

   if a==1
      Pim1d0=posbase;
   else
      Pim1d0=(Pid0~=0).*[pim0;pim1;pim2];
   end

   Pidim1t=(Pidim1~=0).*[pi0;pi1;pi2];
   Pid0=Rim1v0*Pidim1t+Pim1d0;

   % Écriture de Pidim1
   fprintf(h,'\nPidim1=\n');

   for b=1:3

      Pidim1t=[char(Pidim1(b)),';\n'];

      Pidim1t=strrep(Pidim1t,'dii','u[16]');
      Pidim1t=strrep(Pidim1t,'aim1t','u[14]');

      Pidim1t=strrep(Pidim1t,'=0;','=0.0;');
      Pidim1t=strrep(Pidim1t,'=1;','=1.0;');

      fprintf(h,['   y[',num2str(b+21),']=']);
      fprintf(h,Pidim1t);

   end

   % Écriture de Pid0
   fprintf(h,'\nPid0=\n');

   for b=1:3

      Pid0t=[char(Pid0(b)),';\n'];
```

```
        Pid0t=strrep(Pid0t,'r011','u[1]');
        Pid0t=strrep(Pid0t,'r012','u[2]');
        Pid0t=strrep(Pid0t,'r013','u[3]');
        Pid0t=strrep(Pid0t,'r021','u[4]');
        Pid0t=strrep(Pid0t,'r022','u[5]');
        Pid0t=strrep(Pid0t,'r023','u[6]');
        Pid0t=strrep(Pid0t,'r031','u[7]');
        Pid0t=strrep(Pid0t,'r032','u[8]');
        Pid0t=strrep(Pid0t,'r033','u[9]');

        Pid0t=strrep(Pid0t,'pi0','y[22]');
        Pid0t=strrep(Pid0t,'pi1','y[23]');
        Pid0t=strrep(Pid0t,'pi2','y[24]');

        Pid0t=strrep(Pid0t,'pim0','u[10]');
        Pid0t=strrep(Pid0t,'pim1','u[11]');
        Pid0t=strrep(Pid0t,'pim2','u[12]');

        %Pid0t=strrep(Pid0t,'dii','u[17]');
        %Pid0t=strrep(Pid0t,'aim1t','u[15]');

        Pid0t=strrep(Pid0t,'=0;','=0.0;');

        fprintf(h,['    y[',num2str(b+9),']=']);
        fprintf(h,Pid0t);

    end


    fclose(h);

end


% Création des M fichiers source d'algorithmes d'itérations
% de la cinématique | Compilation des fichiers source

disp('Création et compilation des fichiers C-MEX de cinématique');
for a=1:N

    vrainom1=[noms3,num2str(a),'.c'];
    nomeq=[nomc,num2str(a),'.txt'];
    eval(['!perl modsourcecin.pl ',vrainom1,' ',nomeq]); % Création
    eval(['mex ',vrainom1]); %Compilation

end


% Production des librairies contenant les blocs d'appel des
% S-Function de cinématique directe

eval(['!perl prodlibcin.pl ',num2str(N),' ',nomlibcin,'.mdl']);


% Production de la fonction de recombinaison des entrées de
% paramètres, de variables articulaires et de cinématique
```

```
% directe pour la fonction de dynamique inverse articulaire

% Association des sorties aux entrée de la fonction
for a=0:(M-1)
   eval(['y',num2str(a*39),'=[''u[',num2str(a),'];''];']); %qid
   eval(['y',num2str(a*39+1),'=[''u[',num2str(a+M),'];''];']); %qidd
   eval(['y',num2str(a*39+2),'=[''u[',num2str(a+2*M),'];''];']); %masse
   for b=0:2

eval(['y',num2str(a*39+b+3),'=[''u[',num2str(a*3+2*M+N+b),'];''];']);
%pcidi
   end
   for b=0:8

eval(['y',num2str(a*39+b+6),'=[''u[',num2str(a*9+2*M+4*N+b),'];''];']);
%tenseur
   end
   for b=0:8

eval(['y',num2str(a*39+b+15),'=[''u[',num2str(a*9+2*M+13*N+b),'];''];'])
; %Rivim1
   end
   for b=0:8

eval(['y',num2str(a*39+b+24),'=[''u[',num2str((a+1)*9+2*M+13*N+b),'];''']
;']); %Rip1vi
   end
   for b=0:2

eval(['y',num2str(a*39+b+33),'=[''u[',num2str(a*3+2*M+22*N+b),'];''];'])
; %Pidim1
   end
   for b=0:2

eval(['y',num2str(a*39+b+36),'=[''u[',num2str((a+1)*3+2*M+22*N+b),'];''']
;']); %Pip1di
   end
end

% Écriture du fichier des sorties
% Ouverture du fichier de l'articulation M
eval(['h=fopen(''recomb.txt'',''w'');']);

% Écriture de l'entête du fichier
eval(['fprintf(h,''Fichier d''''association des sorties aux
entrées\n\nSorties:\n'');']);
for a=0:(M*39-1)

eval(['fprintf(h,''y[',num2str(a),']=',eval(['y',num2str(a)]),'\n'');'])
;
end

% Fermeture du fichier de l'articulation M
fclose(h);

% Modification de Recombine.c avec recombine.pl (introduction des
sorties) et compilation
```

```
!perl recombine.pl
mex recombine.c

% Modification et compilation du fichier source de la division d'un
vecteur par une matrice
eval(['!perl libconst.pl ',num2str(M)]);
mex divmat.c
```

Coquille de la librairie de blocs d'appel de fonction calculant la cinématique directe (début de la fonction)

```
Library {
  Name                    "libcin"
  Version          2.20
  BlockDefaults {
    Orientation             right
    ForegroundColor         black
    BackgroundColor         white
    DropShadow              off
    NamePlacement     normal
    FontName                "Helvetica"
    FontSize                10
    FontWeight              normal
    FontAngle               normal
    ShowName                on
  }
  AnnotationDefaults {
    HorizontalAlignment     center
    VerticalAlignment       middle
    ForegroundColor         black
    BackgroundColor         white
    DropShadow              off
    FontName                "Helvetica"
    FontSize                10
    FontWeight              normal
    FontAngle               normal
  }
  LineDefaults {
    FontName                "Helvetica"
    FontSize                9
    FontWeight              normal
    FontAngle               normal
  }
  System {
    Name              "libcin"
    Location                [41, 172, 785, 330]
    Open              on
    ToolBar           on
    StatusBar               on
    ScreenColor             white
    PaperOrientation        landscape
    PaperPositionMode       auto
```

```
PaperType               usletter
PaperUnits              inches
```

Coquille de la librairie de blocs d'appel de fonction calculant la cinématique directe (une articulation)

```
Block {
  BlockType              SubSystem
  Name                "cin_dir_art"
  Ports               [3, 3, 0, 0, 0]
  Position               [1,2,3,4]
  ShowPortLabels         on
  System {
  Name                "cin_dir_art"
  Location            [205, 123, 559, 318]
  Open                off
  ToolBar                off
  StatusBar           off
  ScreenColor         white
  PaperOrientation    landscape
  PaperPositionMode   auto
  PaperType           usletter
  PaperUnits          inches
  Block {
    BlockType          Inport
    Name                    "prop_e"
    Position           [40, 48, 70, 62]
    Port                    "1"
    PortWidth          "-1"
    SampleTime              "-1"
  }
  Block {
    BlockType          Inport
    Name                    "qi"
    Position           [40, 93, 70, 107]
    Port                    "2"
    PortWidth          "-1"
    SampleTime              "-1"
  }
  Block {
    BlockType          Inport
    Name                    "param"
    Position           [40, 138, 70, 152]
    Port                    "3"
    PortWidth          "-1"
    SampleTime              "-1"
  }
  Block {
    BlockType          Demux
    Name                    "Demux"
    Ports                   [1, 3, 0, 0, 0]
    Position           [240, 35, 245, 165]
    ShowName           off
    Outputs            "[13 9 3]"
```

```
}
Block {
  BlockType         Mux
  Name                    "Mux"
  Ports                   [3, 1, 0, 0, 0]
  Position          [110, 34, 115, 166]
  ShowName          off
  Inputs            "3"
}
Block {
  BlockType         "S-Function"
  Name                    "S-Function"
  Ports                   [1, 1, 0, 0, 0]
  Position          [145, 85, 205, 115]
  FunctionName            "cindir"
  PortCounts              "[]"
  SFunctionModules        "''"
}
Block {
  BlockType         Outport
  Name                    "prop_s"
  Position          [290, 48, 320, 62]
  Port                    "1"
  OutputWhenDisabled      held
  InitialOutput           "[]"
}
Block {
  BlockType         Outport
  Name                    "Rivim1"
  Position          [290, 93, 320, 107]
  Port                    "2"
  OutputWhenDisabled      held
  InitialOutput           "[]"
}
Block {
  BlockType         Outport
  Name                    "Pidim1"
  Position          [290, 138, 320, 152]
  Port                    "3"
  OutputWhenDisabled      held
  InitialOutput           "[]"
}
Line {
  SrcBlock          "Mux"
  SrcPort           1
  DstBlock          "S-Function"
  DstPort           1
}
Line {
  SrcBlock          "prop_e"
  SrcPort           1
  DstBlock          "Mux"
  DstPort           1
}
Line {
  SrcBlock          "qi"
  SrcPort           1
```

```
        DstBlock          "Mux"
        DstPort           2
      }
      Line {
        SrcBlock          "param"
        SrcPort           1
        DstBlock          "Mux"
        DstPort           3
      }
      Line {
        SrcBlock          "S-Function"
        SrcPort           1
        DstBlock          "Demux"
        DstPort           1
      }
      Line {
        SrcBlock          "Demux"
        SrcPort           1
        DstBlock          "prop_s"
        DstPort           1
      }
      Line {
        SrcBlock          "Demux"
        SrcPort           2
        DstBlock          "Rivim1"
        DstPort           1
      }
      Line {
        SrcBlock          "Demux"
        SrcPort           3
        DstBlock          "Pidim1"
        DstPort           1
      }
      }
    }
```

Coquille de la librairie de blocs d'appel de fonction calculant la dynamique inverse (début de la fonction)

```
Library {
  Name                    "test"
  Version          2.20
  BlockDefaults {
    Orientation             right
    ForegroundColor         black
    BackgroundColor         white
    DropShadow              off
    NamePlacement     normal
    FontName                "Helvetica"
    FontSize                10
    FontWeight              normal
    FontAngle               normal
    ShowName                on
```

```
  }
  AnnotationDefaults {
    HorizontalAlignment     center
    VerticalAlignment       middle
    ForegroundColor         black
    BackgroundColor         white
    DropShadow              off
    FontName                "Helvetica"
    FontSize                10
    FontWeight              normal
    FontAngle               normal
  }
  LineDefaults {
    FontName                "Helvetica"
    FontSize                9
    FontWeight              normal
    FontAngle               normal
  }
  System {
    Name              "test"
    Location                [70, 200, 570, 460]
    Open              on
    ToolBar           on
    StatusBar               on
    ScreenColor             white
    PaperOrientation        landscape
    PaperPositionMode       auto
    PaperType               usletter
    PaperUnits              inches
```

Coquille de la librairie de blocs d'appel de fonction calculant la cinématique directe (une articulation)

```
    Block {
      BlockType               SubSystem
      Name              "Dyn_inv_art"
      Ports             [3, 3, 0, 0, 0]
      Position                [1,2,3,4]
      ShowPortLabels          on
      MaskDisplay       "plot(-
100,0,280,430,[53,42,12,19,72,90,88,54,66"
",95,86,39,32,63,75,83],[0,24,45,80,100,105,149,195,282,312,318,343,382,
399,39"
"3,420],[83,64,85,155,167,137,117,111,80,86,118,137,162,165,118,95,105],
[0,30,"
"36,50,92,127,112,154,201,265,296,287,305,342,373,385,415],[85,67,72],[3
6,64,1"
"00],[155,126,137],[50,77,127],[48,43,54,57,48],[365,376,383,370,365],[3
9,69,6"
"3],[343,362,399],[86,113,118],[318,343,373])"
      MaskIconFrame           on
      MaskIconOpaque          off
```

```
MaskIconRotate          none
MaskIconUnits           autoscale
System {
Name              "Dyn_inv_art"
Location          [196, 203, 698, 540]
Open              off
ToolBar                 off
StatusBar         off
ScreenColor       white
PaperOrientation  landscape
PaperPositionMode auto
PaperType         usletter
PaperUnits        inches
Block {
  BlockType         Inport
  Name                    "bae_e"
  Position          [40, 48, 70, 62]
  Port                    "1"
  PortWidth         "-1"
  SampleTime              "-1"
}
Block {
  BlockType         Inport
  Name                    "q_param"
  Position          [40, 113, 70, 127]
  Port                    "2"
  PortWidth         "-1"
  SampleTime              "-1"
}
Block {
  BlockType         Inport
  Name                    "eab_e"
  Position          [40, 223, 70, 237]
  Port                    "3"
  PortWidth         "-1"
  SampleTime              "-1"
}
Block {
  BlockType         Demux
  Name                    "Demux"
  Ports                   [1, 2, 0, 0, 0]
  Position          [365, 43, 370, 132]
  ShowName          off
  Outputs           "[9 21]"
}
Block {
  BlockType         Demux
  Name                    "Demux1"
  Ports                   [1, 2, 0, 0, 0]
  Position          [365, 208, 370, 297]
  ShowName          off
  Outputs           "[1 6]"
}
Block {
  BlockType         Mux
  Name                    "Mux"
  Ports                   [2, 1, 0, 0, 0]
```

```
  Position          [140, 22, 145, 153]
  ShowName          off
  Inputs            "2"
}
Block {
  BlockType         Mux
  Name                    "Mux1"
  Ports                   [2, 1, 0, 0, 0]
  Position          [140, 206, 145, 299]
  ShowName          off
  Inputs            "[6 21]"
}
Block {
  BlockType         "S-Function"
  Name                    "S-Function"
  Ports                   [1, 1, 0, 0, 0]
  Position          [175, 75, 320, 105]
  FunctionName            "dyninvbae"
  PortCounts              "[]"
  SFunctionModules        "''"
}
Block {
  BlockType         "S-Function"
  Name                    "S-Function1"
  Ports                   [1, 1, 0, 0, 0]
  Position          [175, 240, 320, 270]
  FunctionName            "dyninveab"
  PortCounts              "[]"
  SFunctionModules        "''"
}
Block {
  BlockType         Outport
  Name                    "bae_s"
  Position          [440, 58, 470, 72]
  Port                    "1"
  OutputWhenDisabled      held
  InitialOutput           "0"
}
Block {
  BlockType         Outport
  Name                    "Couple"
  Position          [440, 223, 470, 237]
  Port                    "2"
  OutputWhenDisabled      held
  InitialOutput           "0"
}
Block {
  BlockType         Outport
  Name                    "eab_s"
  Position          [440, 268, 470, 282]
  Port                    "3"
  OutputWhenDisabled      held
  InitialOutput           "0"
}
Line {
  SrcBlock          "Mux"
  SrcPort           1
```

```
  DstBlock          "S-Function"
  DstPort           1
}
Line {
  SrcBlock          "S-Function"
  SrcPort           1
  DstBlock          "Demux"
  DstPort           1
}
Line {
  SrcBlock          "bae_e"
  SrcPort           1
  DstBlock          "Mux"
  DstPort           1
}
Line {
  SrcBlock          "q_param"
  SrcPort           1
  DstBlock          "Mux"
  DstPort           2
}
Line {
  SrcBlock          "Mux1"
  SrcPort           1
  DstBlock          "S-Function1"
  DstPort           1
}
Line {
  SrcBlock          "S-Function1"
  SrcPort           1
  DstBlock          "Demux1"
  DstPort           1
}
Line {
  SrcBlock          "Demux"
  SrcPort           2
  Points            [20, 0; 0, 70; -365, 0; 0, 95]
  DstBlock          "Mux1"
  DstPort           2
}
Line {
  SrcBlock          "eab_e"
  SrcPort           1
  DstBlock          "Mux1"
  DstPort           1
}
Line {
  SrcBlock          "Demux1"
  SrcPort           1
  DstBlock          "Couple"
  DstPort           1
}
Line {
  SrcBlock          "Demux1"
  SrcPort           2
  DstBlock          "eab_s"
  DstPort           1
```

```
      }
      Line {
        SrcBlock          "Demux"
        SrcPort           1
        DstBlock          "bae_s"
        DstPort           1
      }
      }
    }
```

Coquille de la fonction de recombinaison pour la réorganisation des entrées du calcul de la dynamique inverse

```
/*
 * RECOMBINE Fonction de recombinaison des données de la dynamique
inverse
 *
 *
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   UQTR
 *   Copyright (c) aout 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME recombine

#include "simstruc.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 133);/* number of inputs         */
    ssSetNumOutputs(       S, 156);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
```

```
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
   /* Recombinaison_des_entrées */


}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}
```

```
#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Programme Perl d'ajout des équations dans la routine de calcul de la cinématique directe

```
#c:\matlab\bin\perl

# Programme de modification des codes sources de calcul de la
cinématique directe
#
# Entrées : noml : Nom du fichier destination où seront écrits les codes
sources
#                  de calcul de cinématique directe
#           nomeq : Nom du fichier texte contenant les équations à être
écrites
#                   dans les codes sources produits par ce programme
#
#
# Par : Martin de Montigny
# Opal-rt
# Uqtr
# Aout 1998

my($noml, $nomeq);
$noml=$ARGV[0];
$nomeq=$ARGV[1];
if((! $noml)||(! $nomeq))
{
   print "ERREUR (nom incorrect) \n";
}

open(INS,"<$nomeq") or die "fichier source cinématique peut pas être
ouvert!";
open(INCIN,"<cin_dir.c") or die "impossible d'ouvrir cin_dir";
open(OUTCIN,">$noml") or die "impossible d'ouvrir fichier destination";

##########################
## Cinématique directe ##
##########################

while(<INCIN>)
{

   if(/S_FUNCTION_NAME/)
   {
      print OUTCIN "#define S_FUNCTION_NAME $noml\n";
   }
   elsif(/Calcul\s*de\s*trigo:/)
```

```
{
   #Écrire commentaire
   printf OUTCIN $_;

   #Trouver equations de vitesse dans fichier source
   while(<INS>)
   {

      #Tester si on est rendu aux equations consernées
      if(/trigo:/)
      {
         #Récupérer les 5 équations de di et de trigo
         $_=<INS>;
         $di=$_;
         $_=<INS>;
         $sthetai=$_;
         $_=<INS>;
         $cthetai=$_;
         $_=<INS>;
         $salphaiml=$_;
         $_=<INS>;
         $calphaiml=$_;

         printf OUTCIN $di;
         printf OUTCIN $sthetai;
         printf OUTCIN $cthetai;
         printf OUTCIN $salphaim1;
         printf OUTCIN $calphaim1;
      }
   }

}
elsif(/Calcul\s*de\s*Rivim1:/)
{
   #Écrire commentaire
   printf OUTCIN $_;

   #Trouver equations de d'accélération rotative dans fichier source
   seek INS,0,0;
   while(<INS>)
   {

      #Tester si on est rendu aux equations consernées
      if(/Rivim1=/)
      {
         #Récupérer les 9 éléments de la matrice de rotation
         for($i=0;$i<9;$i++)
         {
               $_=<INS>;
               printf OUTCIN $_;
         }

      }
   }
}
elsif(/Calcul\s*de\s*Pidim1:/)
{
```

```
   #Écrire commentaire
   printf OUTCIN $_;

   #Trouver vecteur de position relatif dans fichier source
   seek INS,0,0;
   while(<INS>)
   {

      #Tester si on est rendu aux equations consernées
      if(/Pidiml=/)
      {
         #Récupérer les 3 équations du vecteur de position
         for($i=0;$i<3;$i++)
         {
               $_=<INS>;
               printf OUTCIN $_;
         }

      }
   }
}
elsif(/Calcul\s*de\s*Riv0:/)
{
   #Écrire commentaire
   printf OUTCIN $_;

   #Trouver equations de force dans fichier source
   seek INS,0,0;
   while(<INS>)
   {
      #Tester si on est rendu aux equations consernées
      if(/Riv0=/)
      {
         for($i=0;$i<9;$i++)
         {
               $_=<INS>;
               printf OUTCIN $_;
         }
      }
   }
}

elsif(/Calcul\s*de\s*Pid0:/)
{
   #Écrire commentaire
   printf OUTCIN $_;

   #Trouver equations de force dans fichier source
   seek INS,0,0;
   while(<INS>)
   {
      #Tester si on est rendu aux equations consernées
      if(/Pid0=/)
      {
         for($i=0;$i<3;$i++)
         {
               $_=<INS>;
```

```
                printf OUTCIN $_;
            }
         }
      }
   }

   else
   {
      printf OUTCIN $_;
   }
}


close(INS);
close(INCIN);
close(OUTCIN);
```

Programme Perl d'ajout des équations dans la routine de calcul de la dynamique inverse

```
#c:\matlab\bin\perl

# Programme de modification des codes sources de calcul de la dynamique
inverse
#
# Entrées : nom1 : nom du fichier créé pour le calcul de la dynamique
inverse,
#                  itération de la base à l'effecteur
#           nom2 : nom du fichier créé pour le calcul de la dynamique
inverse,
#                  itération de l'effecteur à la base
#           nomeq : nom du fichier texte contenant les équations à être
écrites
#                   dans les codes sources produits par ce programme
#
# Par : Martin de Montigny
# Opal-rt
# Uqtr
# aout 1998

my($nom1, $nom2, $nomeq);
$nom1=$ARGV[0];
$nom2=$ARGV[1];
$nomeq=$ARGV[2];
if((! $nom1)||(! $nom2)||(! $nomeq))
{
   print "ERREUR (nom incorrect) \n";
}

open(INS,"<$nomeq") or die "doigtne1 peut pas etre ouvert!";
open(INBAE,"<dyninv_bae.c") or die "impossible d'ouvrir dyninv_bae";
open(OUTBAE,">$nom1") or die "impossible d'ouvrir dyninvbae";
open(INEAB,"<dyninv_eab.c") or die "impossible d'ouvrir dyninv_eab";
```

```
open(OUTEAB,">$nom2") or die "impossible d'ouvrir dyninveab";

########################################
## Itération de la base à l'effecteur ##
########################################

while(<INBAE>)
{

   if(/vitesse\s*de\s*rotation/)
   {
      #Écrire commentaire
      printf OUTBAE $_;

      #Trouver equations de vitesse dans fichier source
      while(<INS>)
      {

         #Tester si on est rendu aux equations consernées
         if(/widi=/)
         {
            #Récupérer les 3 équations de vitesse rotative
            $_=<INS>;
            $widia=$_;
            $_=<INS>;
            $widib=$_;
            $_=<INS>;
            $widic=$_;

            printf OUTBAE $widia;
            printf OUTBAE $widib;
            printf OUTBAE $widic;
         }
      }

   }
   elsif(/acceleration\s*rotative/)
   {
      #Écrire commentaire
      printf OUTBAE $_;

      #Trouver equations de d'accélération rotative dans fichier source
      seek INS,0,0;
      while(<INS>)
      {

         #Tester si on est rendu aux equations consernées
         if(/wpidi=/)
         {
            #Récupérer les 3 équations de d'accélération rotative
            $_=<INS>;
            $wpidia=$_;
            $_=<INS>;
            $wpidib=$_;
            $_=<INS>;
            $wpidic=$_;
```

```
            printf OUTBAE $wpidia;
            printf OUTBAE $wpidib;
            printf OUTBAE $wpidic;
         }
      }
   }
   elsif(/acceleration\s*lineaire/)
   {
      #Écrire commentaire
      printf OUTBAE $_;

      #Trouver equations de d'accélération linéaire dans fichier source
      seek INS,0,0;
      while(<INS>)
      {

         #Tester si on est rendu aux equations consernées
         if(/vpidi=/)
         {
            #Récupérer les 3 équations d'accélération linéaire
            $_=<INS>;
            $vpidia=$_;
            $_=<INS>;
            $vpidib=$_;
            $_=<INS>;
            $vpidic=$_;

            printf OUTBAE $vpidia;
            printf OUTBAE $vpidib;
            printf OUTBAE $vpidic;
         }
      }
   }
   elsif(/forces\s*et\s*couples\s*inertiels/)
   {
      #Écrire commentaire
      printf OUTBAE $_;

      #Trouver equations de force dans fichier source
      seek INS,0,0;
      while(<INS>)
      {
         #Tester si on est rendu aux equations consernées
         if(/Fi=/)
         {
            #Récupérer les 6 équations de force et de couple
            $_=<INS>;
            $Fia=$_;
            $_=<INS>;
            $Fib=$_;
            $_=<INS>;
            $Fic=$_;
            $_=<INS>;
            $_=<INS>;
            $_=<INS>;
            $Nia=$_;
            $_=<INS>;
```

```
            $Nib=$_;
            $_=<INS>;
            $Nic=$_;

            printf OUTBAE $Fia;
            printf OUTBAE $Fib;
            printf OUTBAE $Fic;
            printf OUTBAE $Nia;
            printf OUTBAE $Nib;
            printf OUTBAE $Nic;
         }
      }
   }

   else
   {
      printf OUTBAE $_;
   }
}

########################################
## Itération de l'effecteur à la base ##
########################################

while(<INEAB>)
{

   if(/forces\s*et\s*couples\s*externes\*/)
   {
      #Écrire commentaire
      printf OUTEAB $_;

      #Trouver equations de vitesse dans fichier source
      seek INS,0,0;
      while(<INS>)
      {

         #Tester si on est rendu aux equations consernées
         if(/fi=/)
         {
            #Récupérer les 6 équations de vitesse rotative
            $_=<INS>;
            $fia=$_;
            $_=<INS>;
            $fib=$_;
            $_=<INS>;
            $fic=$_;
            $_=<INS>;
            $_=<INS>;
            $_=<INS>;
            $nia=$_;
            $_=<INS>;
            $nib=$_;
            $_=<INS>;
            $nic=$_;

            printf OUTEAB $fia;
```

```
            printf OUTEAB $fib;
            printf OUTEAB $fic;
            printf OUTEAB $nia;
            printf OUTEAB $nib;
            printf OUTEAB $nic;
         }
      }
   }
   elsif(/couple\s*articulaire\*/)
   {
      #Écrire commentaire
      printf OUTEAB $_;

      #Trouver equations de vitesse dans fichier source
      seek INS,0,0;
      while(<INS>)
      {

         #Tester si on est rendu aux equations consernées
         if(/Ti=/)
         {
            #Récupérer l'équation de couple
            $_=<INS>;
            $Ti=$_;
            printf OUTEAB "   y[0]=";
            printf OUTEAB $Ti;
         }
      }
   }
   else
   {
      printf OUTEAB $_;
   }
}

close(INS);
close(INBAE);
close(OUTBAE);
close(INEAB);
close(OUTEAB);
```

Programme Perl de production de la librairie des blocs d'appel des routines calculant la cinématique directe

```
#!c:\matlab\bin\perl

# Programme de production de la librairie de blocs d'appel des "s-
functions"
# de calcul de la cinématique directe
#
# Entrées : N : Nombre d'articulations total du manipulateur
#           nomlib : nom de la librairie produite par ce programme
#
```

```
# Par : Martin de Montigny
# Opal-rt
# Uqtr
# Aout 1998

my($N, $nomlib);

$N = $ARGV[0];
$nomlib = $ARGV[1];
if((! $N)||(! $nomlib))
{
    print "ERREUR : mauvais passage de paramètres.\n";
}

open(DEBUTS,"<debutlibcin.txt") or die "Impossible d'ouvrir
debutlib.txt";
open(BLOCKS,"<bloclibcin.txt") or die "Impossible d'ouvrir
blocklib.txt";
open(LIB,">$nomlib") or die "Cannot create $nomlib";

# Écrire le début de la librairie (partie constante)
while(<DEBUTS>)
{
   print LIB $_;
}

# Écriture des blocs nécessaires au manipulateur actuel
for($i=1;$i<=$N;$i++)
{
   seek BLOCKS,0,0;
   while(<BLOCKS>)
   {

      if(/cin_dir_art/)
      {
         print LIB "Name              \"cin_dir_art$i\"\n";
      }
      elsif(/\[1,2,3,4\]/)
      {
         $xmin=45+120*($i-1);
         $xmax=130+120*($i-1);
         $ymin=51;
         $ymax=99;
         print LIB "  Position            [$xmin, $ymin, $xmax,
$ymax]\n";
      }
      elsif(/cindir/)
      {
         print LIB "    FunctionName        \"cindir$i\"\n";
      }
      else
      {
         print LIB $_;
      }
   }
   print LIB "\n";
}
```

```
# Écriture de la fin de la librairie (partie constante)
print LIB "  \}\n\}\n";


close(DEBUTS);
close(BLOCKS);
close(LIB);
```

Programme Perl de production de la librairie des blocs d'appel des routines calculant la dynamique inverse

```
#!c:\matlab\bin\perl

# Programme de production des librairies de blocs d'appel des "s-
functions"
# de calcul de la Dynamique inverse
#
# Entrées : N : Nombre d'articulations actives du manipulateur
#           nomlib : nom de la librairie produite par ce programme
#
# Par : Martin de Montigny
# Opal-rt
# Uqtr
# 1998-99

my($N, $nomlib, $numlib);

$N = $ARGV[0];
$nomlib = $ARGV[1];

if((! $N)||(! $nomlib))
{
    print "ERREUR : mauvais passage de paramètres.\n";
}

open(DEBUTS,"<debutlibne.txt") or die "Impossible d'ouvrir
debutlib.txt";
open(BLOCKS,"<bloclibne.txt") or die "Impossible d'ouvrir blocklib.txt";
open(LIB,">$nomlib") or die "Cannot create $nomlib";

# Écrire le début de la librairie (partie constante)
while(<DEBUTS>)
{
   print LIB $_;
}

# Écriture des blocs nécessaires au manipulateur actuel
for($i=1;$i<=$N;$i++)
{
   seek BLOCKS,0,0;
   while(<BLOCKS>)
   {
```

```
      if(/Dyn_inv_art/)
      {
         print LIB "Name                \"Dyn_inv_art$i\"\n";
      }
      elsif(/\[1,2,3,4\]/)
      {
         $xmin=40+140*($i-1);
         $xmax=145+140*($i-1);
         $ymin=32;
         $ymax=108;
         print LIB "  Position              [$xmin, $ymin, $xmax,
$ymax]\n";
      }
      elsif(/dyninvbae/)
      {
         print LIB "    FunctionName          \"dyninvbae$i\"\n";
      }
      elsif(/dyninveab/)
      {
         print LIB "    FunctionName          \"dyninveab$i\"\n";
      }
      else
      {
         print LIB $_;
      }
   }
   print LIB "\n";
}

# Écriture de la fin de la librairie
print LIB "  \}\n\}\n";


close(DEBUTS);
close(BLOCKS);
close(LIB);
```

Programme Matlab de transformation des puissances en multiplications répétées

```
function [sor]=transpm(ent);
% Transpm.m
%
% Cette fonction transforme les opérations de puissance au carré en
opérations
% de multiplications dans un vacteur symbolique.
%
% Par : Maratin de Montigny
% UQTR, Opal-rt
% 1998

sent=length(ent);
```

```
syms sor real
sor=sym(zeros(length(ent),1));

for a=1:sent

   % Élément présent à transformer
   entt=ent(a);
   ents=strvcat(entt); % version string de l'élément...

   % Vecteur de positions de "^2"
   posp2=findstr(ents,'^2');

   % Nombre de transformations requises
   nt=length(posp2);

   for b=1:nt

      % Vecteur de positions de "^2" (recalcule car "ents" change...)
      posp2=findstr(ents,'^2');

      % Trouver position avant début de la variable ^2
      pav=posp2(1); % Initialisation à la position de l'exposant
      while
(ents(pav)~='='&ents(pav)~='('&ents(pav)~='+'&ents(pav)~='*'& ...
            ents(pav)~='/'&ents(pav)~='-')
         pav=pav-1;
      end

      % Trouver position après fin de la variable ^2
      pap=posp2(1)+2;

      % Modifier la chaine de char représentant l'élément "a" du vecteur
      ents=[ents(1:pav),ents(pav+1:posp2(1)-1),'*',ents(pav+1:posp2(1)-
1),ents(pap:length(ents))];

   end

   sor(a,1)=ents;
end
```

Annexe B

Programme Matlab de gestion de l'interface de l'utilisateur

```
function
[Gravdir,ki,alphaim1,aim1,di,thetai,masse,ccmdh,typten,param,tenseur,n,
...

nomnou,N,M,fv,rbase,rotbase,posbase,propcin,paramcin,Prop_ne_vit,Prop_ne
_mas,ctes,pcidi]= ...

finitr(choix,Gravdir,ki,alphaim1,aim1,di,thetai,masse,ccmdh,typten,param
, ...

tenseur,n,nomnou,N,M,fv,rbase,rotbase,posbase,propcin,paramcin,Prop_ne_v
it,Prop_ne_mas,ctes,pcidi);
% finitr.m
%
% Cette fonction sert à effectuer la gestion de fichier pour le
% menu d'initialisation des paramètres de la simulation du
% manipulateur robotique. (Lagrange-Euler)
%
% Par : Martin de Montigny,   1998-1999
%
% Les foncitons disponibles sont les suivantes :
% 1) Créer un nouveau fichier
% 2) Ouvrir un nouveau fichier
% 3) Sauver un fichier
% 4) Aide en ligne
% 5) Quitter sans sauver
% 6) Rafraichissement général des fenêtres...
% 7) Créer les paramètres (à zéro) (le ok du menu créer...)
% 8) Annuler (fermer la fenêtre courante)
% 9) OK (de la fenêtre de l'option ouvrir)
% 10) Active tous les boutons (1 fois que l'utilisateur a
%     chargé ou créé le fichier de paramètres.
% 20) Génération automatique de code par la formulation de Lagrange


if choix==1

   load initrfig mat0 mat1 mat2

   clear flag1 flag2 flag3
    % Jusqu'à temps que l'utilisateur ait entré un nom
    % et un nombre d'articulations N et M (totales + actives)

   a = figure('Color',[0 0.1 0.4], ...
         'Colormap',mat0, ...
         'PointerShapeCData',mat1, ...
         'Position',[105 221 460 387], ...
         'Tag','Fig111');
```

```
   b = uicontrol('Parent',a, ...
         'Units','points', ...
         'BackgroundColor',[0 0.1 0.4], ...
         'FontSize',12, ...
         'ForegroundColor',[0.2 1 1], ...
         'Position',[38 181 215 45], ...
         'String','Quel nom voulez-vous donner à votre fichier de
paramètres ?', ...
         'Style','text', ...
         'Tag','StaticText111');
   c = uicontrol('Parent',a, ...
         'Units','points', ...
      'BackgroundColor',[0.1 0.2 0.5], ...
         'ForegroundColor',[0.2 1 1], ...
      'FontSize',12, ...
         'Callback','flag1=1;nomnou=get(gcbo,''String'');if
exist(''flag1'')&exist(''flag2'')&exist(''flag3'')
set(findobj(''Tag'',''Pushbutton111''),''Enable'',''on'');end;', ...
         'Position',[88 147 124 30], ...
         'Style','edit', ...
         'Tag','EditText111');
   d = uicontrol('Parent',a, ...
         'Units','points', ...
         'BackgroundColor',[0.1 0.2 0.5], ...
         'ForegroundColor',[0.2 1 1], ...
         'Position',[160 106 52 17], ...
         'String',mat2, ...
         'Style','popupmenu', ...
      'Tag','PopupMenu111', ...
      'Callback','flag2=1;N=get(gcbo,''Value'');if
exist(''flag1'')&exist(''flag2'')&exist(''flag3'')
set(findobj(''Tag'',''Pushbutton111''),''Enable'',''on'');end;', ...
      'Value',1);
   dd = uicontrol('Parent',a, ...
         'Units','points', ...
         'BackgroundColor',[0.1 0.2 0.5], ...
         'ForegroundColor',[0.2 1 1], ...
         'Position',[160 72 52 17], ...
         'String',mat2, ...
         'Style','popupmenu', ...
      'Tag','PopupMenu111', ...
      'Callback','flag3=1;M=get(gcbo,''Value'');if
exist(''flag1'')&exist(''flag2'')&exist(''flag3'')
set(findobj(''Tag'',''Pushbutton111''),''Enable'',''on'');end;', ...
      'Value',1);
   e = uicontrol('Parent',a, ...
         'Units','points', ...
         'BackgroundColor',[0 0.1 0.4], ...
         'ForegroundColor',[0.2 1 1], ...
         'Position',[67 106 85 15], ...
         'String','Nombre d''articulations', ...
         'Style','text', ...
      'Tag','StaticText111');
   ee = uicontrol('Parent',a, ...
         'Units','points', ...
         'BackgroundColor',[0 0.1 0.4], ...
         'ForegroundColor',[0.2 1 1], ...
```

```
          'Position',[35 71 117 16], ...
          'String','Nombre d''articulations actives', ...
          'Style','text', ...
       'Tag','StaticText111');
   f = uicontrol('Parent',a, ...
          'Units','points', ...
          'Callback','clear flag1 flag2
flag3;[Gravdir,ki,alphaim1,aim1,di,thetai,masse,ccmdh,typten,param,tense
ur,n,nomnou,N,M,fv,rbase,rotbase,posbase,propcin,paramcin,Prop_ne_vit,Pr
op_ne_mas,ctes,pcidi]=finitr(7,N,M);', ...
          'FontSize',12, ...
          'Position',[102 20 91 33], ...
       'String','O K', ...
       'Enable','off', ...
       'Tag','Pushbutton111');

elseif choix==2

   load initrfig mat3 mat4

   % Va chercher la liste des fichiers .mat
   a=dir('*.mat');
   b=strvcat(a.name);

   a = figure('Color',[0 0.1 0.4], ...
          'Colormap',mat3, ...
          'PointerShapeCData',mat4, ...
          'Position',[436 206 405 330], ...
          'Tag','Fig1');
   uicontrol('Parent',a, ...
          'Units','points', ...
          'BackgroundColor',[0 0.1 0.4], ...
          'FontSize',12, ...
          'ForegroundColor',[0.2 1 1], ...
          'Position',[19.8621 173.172 214.138 22.3448], ...
          'String','Quel est le nom du fichier à ouvrir ?', ...
          'Style','text', ...
          'Tag','StaticText111');
   uicontrol('Parent',a, ...
          'Units','points', ...

'Callback','[Gravdir,ki,alphaim1,aim1,di,thetai,masse,ccmdh,typten,param
,tenseur,n,nomnou,N,M,fv,rbase,rotbase,posbase,propcin,paramcin,Prop_ne_
vit,Prop_ne_mas,ctes,pcidi]=finitr(9);', ...
          'FontSize',12, ...
          'Position',[139.655 18.6207 90.6207 32.8966], ...
          'String','O K', ...
          'Tag','Pushbutton111');
   uicontrol('Parent',a, ...
          'Units','points', ...
          'Callback','finitr(8);', ...
          'FontSize',12, ...
          'Position',[32.2759 19.2414 90.6207 32.8966], ...
          'String','ANNULER', ...
          'Tag','Pushbutton111');
   uicontrol('Parent',a, ...
          'Units','points', ...
```

```
        'BackgroundColor',[0.1 0.2 0.7], ...
        'ForegroundColor',[0.2 1 1], ...
        'Position',[32.2759 73.8621 194.897 91.8621], ...
        'String',b, ...
        'Style','listbox', ...
        'Tag','Listbox111', ...
        'Value',1);

elseif choix==3

   % Sauvegarde des données
   eval(['save ',nomnou, ...
         ' Gravdir ki alphaim1 aim1 di thetai masse ccmdh typten param
tenseur n nomnou N M fv rbase rotbase posbase propcin paramcin
Prop_ne_vit Prop_ne_mas ctes pcidi']);

elseif choix==4

   % Aide en ligne
   emplacement=Gravdir; %chemin d'accès avant le répertoire aide...
   eval(['web file:///',emplacement,'\aide\aide.htm']);


elseif choix==5

   % Quitter
   finitr(8);

elseif choix==6

   % Rafraichissement général
   h=findobj('Tag','EditText1');

set(h,'String',['[',num2str(Gravdir(1,1)),';',num2str(Gravdir(2,1)),';',
num2str(Gravdir(3,1)),']']);
   h=findobj('Tag','PopupMenu3');
   set(h,'Value',(ki(n,1)+1));
   h=findobj('Tag','EditText3');
   set(h,'String',num2str(alphaim1(n,1)));
   h=findobj('Tag','EditText4');
   set(h,'String',num2str(aim1(n,1)));
   h=findobj('Tag','EditText5');
   set(h,'String',num2str(di(n,1)));
   h=findobj('Tag','EditText6');
   set(h,'String',num2str(thetai(n,1)));
   h=findobj('Tag','EditText7');
   set(h,'String',num2str(masse(n,1)));
   h=findobj('Tag','EditText8');

set(h,'String',['[',num2str(ccmdh(n,1)),';',num2str(ccmdh(n,2)),';',num2
str(ccmdh(n,3)),';',num2str(ccmdh(n,4)),']']);
   h=findobj('Tag','EditText1000');
   set(h,'String',['[',num2str(pcidi((n-1)*3+1,1)),';',num2str(pcidi((n-
1)*3+2,1)),';',num2str(pcidi((n-1)*3+3,1)),']']);
   h=findobj('Tag','EditText21');
   set(h,'String',num2str(fv(n,1)));
```

```
   % Calcul de la matrice de rotation de la base
   rx=rbase(1,1);
   ry=rbase(2,1);
   rz=rbase(3,1);
   rotbase = [cos(ry)*cos(rz);-cos(ry)*sin(rz);sin(ry); ...
         sin(rx)*sin(ry)*cos(rz)+cos(rx)*sin(rz);-
sin(rx)*sin(ry)*sin(rz)+ ...
         cos(rx)*cos(rz);-sin(rx)*cos(ry);-
cos(rx)*sin(ry)*cos(rz)+sin(rx)*sin(rz); ...
         cos(rx)*sin(ry)*sin(rz)+sin(rx)*cos(rz);cos(rx)*cos(ry)];
   clear rx ry rz;

   h=findobj('Tag','EditText22');
   set(h,'String',['[',num2str(rbase(1,1)),';',num2str(rbase(2,1)), ...
         ';',num2str(rbase(3,1)),']']);
   h=findobj('Tag','EditText23');
   set(h,'String',['[',num2str(posbase(1,1)),';',num2str(posbase(2,1)),
...
         ';',num2str(posbase(3,1)),']']);

   % Type de tenseur
   h=findobj('Tag','PopupMenu2');
   set(h,'Value',typten(n,1));

   % Écriture du tenseur s'il est autre que général
   if typten(n,1)==1
      tenseur((n-1)*9+1:(n-1)*9+9,1)=zeros(9,1);

      %cache les paramètres du tenseur
      h=findobj('Tag','StaticText2');
      set(h,'String','');
      set(h,'Enable','off');
      h=findobj('Tag','StaticText3');
      set(h,'String','');
      set(h,'Enable','off');
      h=findobj('Tag','StaticText4');
      set(h,'String','');
      set(h,'Enable','off');
      h=findobj('Tag','EditText9');
      set(h,'String','0');
      set(h,'Enable','off');
      h=findobj('Tag','EditText10');
      set(h,'String','0');
      set(h,'Enable','off');
      h=findobj('Tag','EditText11');
      set(h,'String','0');
      set(h,'Enable','off');


   elseif typten(n,1)==2
      tenseur((n-1)*9+1:(n-
1)*9+9,1)=[masse(n,1)*(3*param(n,1)^2+4*param(n,2)^2)/12; ...
         0;0;0;masse(n,1)*(3*param(n,1)^2+4*param(n,2)^2)/12;0;0;0; ...
         masse(n,1)*param(n,1)^2/2];

      %Affiche les paramètres du tenseur
      h=findobj('Tag','StaticText2');
```

```
      set(h,'Enable','on');
      set(h,'String','Rayon');
      h=findobj('Tag','StaticText3');
      set(h,'Enable','on');
      set(h,'String','Hauteur');
      h=findobj('Tag','StaticText4');
      set(h,'String','');
      set(h,'Enable','off');
      h=findobj('Tag','EditText9');
      set(h,'Enable','on');
      set(h,'String',num2str(param(n,1)));
      h=findobj('Tag','EditText10');
      set(h,'Enable','on');
      set(h,'String',num2str(param(n,2)));
      h=findobj('Tag','EditText11');
      set(h,'String','0');
      set(h,'Enable','off');


   elseif typten(n,1)==3
      tenseur((n-1)*9+1:(n-
1)*9+9,1)=[masse(n,1)*(3*param(n,1)^2+2*param(n,2)^2)/6; ...
         0;0;0;masse(n,1)*(3*param(n,1)^2+2*param(n,2)^2)/6;0;0;0; ...
         masse(n,1)*param(n,1)^2];

      %Affiche les paramètres du tenseur
      h=findobj('Tag','StaticText2');
      set(h,'Enable','on');
      set(h,'String','Rayon');
      h=findobj('Tag','StaticText3');
      set(h,'Enable','on');
      set(h,'String','Hauteur');
      h=findobj('Tag','StaticText4');
      set(h,'String','');
      set(h,'Enable','off');
      h=findobj('Tag','EditText9');
      set(h,'Enable','on');
      set(h,'String',num2str(param(n,1)));
      h=findobj('Tag','EditText10');
      set(h,'Enable','on');
      set(h,'String',num2str(param(n,2)));
      h=findobj('Tag','EditText11');
      set(h,'String','0');
      set(h,'Enable','off');


   elseif typten(n,1)==4
      tenseur((n-1)*9+1:(n-
1)*9+9,1)=[masse(n,1)*(param(n,2)^2+4*param(n,3)^2)/12; ...
         0;0;0;masse(n,1)*(param(n,1)^2+4*param(n,3)^2)/12;0;0;0; ...
         masse(n,1)*(param(n,1)^2+param(n,2)^2)/12];

      %Affiche les paramètres du tenseur
      h=findobj('Tag','StaticText2');
      set(h,'Enable','on');
      set(h,'String','a');
      h=findobj('Tag','StaticText3');
```

```
      set(h,'Enable','on');
      set(h,'String','b');
      h=findobj('Tag','StaticText4');
      set(h,'Enable','on');
      set(h,'String','c');
      h=findobj('Tag','EditText9');
      set(h,'Enable','on');
      set(h,'String',num2str(param(n,1)));
      h=findobj('Tag','EditText10');
      set(h,'Enable','on');
      set(h,'String',num2str(param(n,2)));
      h=findobj('Tag','EditText11');
      set(h,'Enable','on');
      set(h,'String',num2str(param(n,3)));

   end

   %Rafraichissement du tenseur
   h=findobj('Tag','EditText12');
   set(h,'String',num2str(tenseur((n-1)*9+1,1)));
   h=findobj('Tag','EditText13');
   set(h,'String',num2str(tenseur((n-1)*9+2,1)));
   h=findobj('Tag','EditText14');
   set(h,'String',num2str(tenseur((n-1)*9+3,1)));
   h=findobj('Tag','EditText15');
   set(h,'String',num2str(tenseur((n-1)*9+4,1)));
   h=findobj('Tag','EditText16');
   set(h,'String',num2str(tenseur((n-1)*9+5,1)));
   h=findobj('Tag','EditText17');
   set(h,'String',num2str(tenseur((n-1)*9+6,1)));
   h=findobj('Tag','EditText18');
   set(h,'String',num2str(tenseur((n-1)*9+7,1)));
   h=findobj('Tag','EditText19');
   set(h,'String',num2str(tenseur((n-1)*9+8,1)));
   h=findobj('Tag','EditText20');
   set(h,'String',num2str(tenseur((n-1)*9+9,1)));

   % Propagation initiale de la cinématique et paramètres DHmod
   % écrits de façon optimisée pour la simulation
   propcin=[1;rotbase;posbase];
   paramcin=zeros(N-M,1);
   for a=1:N
      paramcin=[paramcin;[aim1(N+1-a);alphaim1(N+1-a);di(N+1-
a);thetai(N+1-a)]];
   end

   % Propagation initiale de la dynamique inverse
   % écrits de façon optimisée pour la simulation
   Prop_ne_vit=[zeros(6,1);-Gravdir]; %L'accélération linéaire est égale
à
                                      % -Gravdir car "Gravdir" indique
la
                                      % direction de la gravité et non
                                      % l'accélération nécessaire pour
                                      % la simuler...
   Prop_ne_mas=zeros(9,1); %Pas de gravité dans le calcul de la mat. des
masses
```

```
   ctes=ctes; % constante à passer pour les calculs de dyn inv le
   fv=fv;

elseif choix==7

   if exist('N')==0
      N=Gravdir; %passage du nombre de paramètres...
      M=ki;
   else
      N=N;
      M=M;
   end

   %Limitation du menu de choix
d'articulation!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   chaine=[];
   for z=1:N
      chaine=[chaine;num2str(z)];
   end
   chaine=cellstr(chaine);
   h=findobj('Tag','PopupMenu1');
   set(h,'String',chaine);

   %Créer les vecteurs de paramètres ayant la longueur spécifiée
   Gravdir=[0;0;-9.81];
%??????????????????????????!!!!!!!!!!!!!!!!!!!!changer pour que Gravdir
soit la DIRECTION de la grav. et non l'opposé...
   ki=zeros(N,1);
   alphaim1=zeros(N,1);
   aim1=zeros(N,1);
   di=zeros(N,1);
   thetai=zeros(N,1);
   masse=zeros(N,1);
   ccmdh=zeros(N,4);
   pcidi=zeros(N*3,1);
   typten=ones(N,1);
   param=zeros(N,3);
   tenseur=zeros(N*9,1);
   fv=zeros(N,1); %M de long en réalité mais problème de dimension...
   rbase=zeros(3,1);
   rotbase=[1;0;0;0;1;0;0;0;1];
   posbase=zeros(3,1);

   %n pointe à la première articulation
   n=1;

   %Sauvegarde du nom...
   h=findobj('Tag','EditText111');
   nomnou=get(h,'String');

   % Fermer la question
   close(gcbf);

   % Active tous les boutons pour permettre modification des paramètres
   finitr(10);
```

```
   % Pour éviter les warnings...
   nomnou=nomnou;

   % Paramètres de la cinématique pour la simulation
   paramcin=zeros(N-M+4*N,1);
   propcin=[1;rotbase;posbase];

   % Constantes pour la simulation de la dynamique directe lagrange
    ctes='';

   % Propagation initiale de la dynamique inverse
   % écrits de façon optimisée pour la simulation
   Prop_ne_vit=[zeros(6,1);-Gravdir]; %L'accélération linéaire est égale
à
                                      % -Gravdir car "Gravdir" indique
la
                                      % direction de la gravité et non
                                      % l'accélération nécessaire pour
                                      % la simuler...
   Prop_ne_mas=zeros(9,1); %Pas de gravité dans le calcul de la mat. des
masses

   %ctes=ctes; % constante à passer pour les calculs de dyn inv le
   fv=fv;

   % Rafraichissement général...

[Gravdir,ki,alphaim1,ccmdh,di,thetai,masse,ccmdh,typten,param,tenseur,n,
nomnou,N,M, ...

fv,rbase,rotbase,posbase,propcin,paramcin,Prop_ne_vit,Prop_ne_mas,ctes,p
cidi]=finitr(6, ...

Gravdir,ki,alphaiml,aiml,di,thetai,masse,ccmdh,typten,param,tenseur,n,no
mnou, ...

N,M,fv,rbase,rotbase,posbase,propcin,paramcin,Prop_ne_vit,Prop_ne_mas,ct
es,pcidi);

elseif choix==8

   %Annuler (fermer la fenêtre)

   close (gcbf)

elseif choix==9

   %OK (ouvrir)
   h=findobj('Tag','Listboxlll');
   pos=get(h,'Value');
   noms=get(h,'String');
   nomnou=noms(pos,:);
   eval(['load ',nomnou,' Gravdir ki alphaiml aiml di thetai masse ccmdh
typten param tenseur n nomnou N M fv rbase rotbase posbase propcin
paramcin Prop_ne_vit Prop_ne_mas ctes pcidi']);

   %Limitation du menu de choix d'articulation
```

```
    chaine=[];
    for z=1:N
       chaine=[chaine;num2str(z)];
    end
    chaine=cellstr(chaine);
    h=findobj('Tag','PopupMenu1');
    set(h,'String',chaine);


    close (gcbf)

    % Active tous les boutons pour permettre modification des paramètres
    finitr(10);

    % Initialise l'articulation présentement observée à 1
    n=1;

    % Rafraichissement général...

[Gravdir,ki,alphaim1,aim1,di,thetai,masse,ccmdh,typten,param,tenseur,n,n
omnou,N,M,fv,rbase,rotbase,posbase,propcin,paramcin,Prop_ne_vit,Prop_ne_
mas,ctes,pcidi]=finitr(6, ...

Gravdir,ki,alphaim1,aim1,di,thetai,masse,ccmdh,typten,param,tenseur,n,no
mnou,N,M,fv,rbase,rotbase,posbase,propcin,paramcin,Prop_ne_vit,Prop_ne_m
as,ctes,pcidi);

elseif choix==10

    % Activer tous les boutons de la fenêtre principale
    h=findobj('Tag','Pushbutton3');
    set(h,'Enable','on');
    h=findobj('Tag','PopupMenu1');
    set(h,'Enable','on');
    h=findobj('Tag','EditText1');
    set(h,'Enable','on');
    h=findobj('Tag','PopupMenu3');
    set(h,'Enable','on');
    h=findobj('Tag','EditText3');
    set(h,'Enable','on');
    h=findobj('Tag','EditText4');
    set(h,'Enable','on');
    h=findobj('Tag','EditText5');
    set(h,'Enable','on');
    h=findobj('Tag','EditText6');
    set(h,'Enable','on');
    h=findobj('Tag','EditText7');
    set(h,'Enable','on');
    h=findobj('Tag','EditText8');
    set(h,'Enable','on');
    h=findobj('Tag','EditText1000');
    set(h,'Enable','on');

    h=findobj('Tag','PopupMenu2');
    set(h,'Enable','on');

    h=findobj('Tag','EditText12');
```

```
   set(h,'Enable','on');
   h=findobj('Tag','EditText13');
   set(h,'Enable','on');
   h=findobj('Tag','EditText14');
   set(h,'Enable','on');
   h=findobj('Tag','EditText15');
   set(h,'Enable','on');
   h=findobj('Tag','EditText16');
   set(h,'Enable','on');
   h=findobj('Tag','EditText17');
   set(h,'Enable','on');
   h=findobj('Tag','EditText18');
   set(h,'Enable','on');
   h=findobj('Tag','EditText19');
   set(h,'Enable','on');
   h=findobj('Tag','EditText20');
   set(h,'Enable','on');
   h=findobj('Tag','EditText21');
   set(h,'Enable','on');
   h=findobj('Tag','EditText22');
   set(h,'Enable','on');
   h=findobj('Tag','EditText23');
   set(h,'Enable','on');
   h=findobj('Tag','Pop1000');
   set(h,'Enable','on');


elseif choix==20 %génération automatique de code

   % Test pour savoir quel formulation utiliser (LE ou NE)
   h=findobj('Tag','Pop1000');
   if (get(h,'Value')==1) % Lagrange-Euler

[Gravdir,ki,alphaim1,aiml,di,thetai,masse,ccmdh,typten,param,tenseur,n,
...

nomnou,N,M,fv,rbase,rotbase,posbase,propcin,paramcin,Prop_ne_vit,Prop_ne
_mas,ctes,pcidi]= ...

genalgo_le(choix,Gravdir,ki,alphaim1,aim1,di,thetai,masse,ccmdh,typten,p
aram, ...

tenseur,n,nomnou,N,M,fv,rbase,rotbase,posbase,propcin,paramcin,Prop_ne_v
it,Prop_ne_mas,ctes,pcidi);
      %load constantes_masses
   else % Newton-Euler
      cd ..
      cd librairie
      genalgo_ne
   end

end
```

Annexe C

Librairie de calcul matriciel et robotique

Calcul matriciel

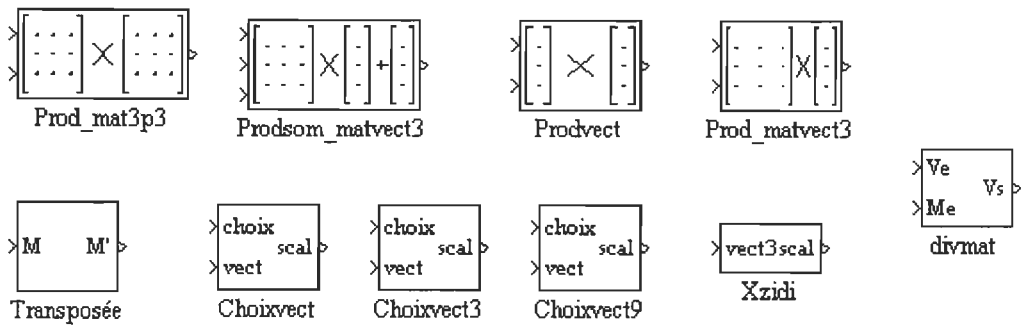

Figure C. 1 : Librairie de calcul matriciel

Les fonctions appelées par ces blocs sont présentés ci-dessous.

Produit de deux matrices 3 x 3 (Prod_mat3p3)

```
/*
 * PROD2MAT3P3 Produit de 2 matrices 3 X 3 dans Simulink
 *
 * Entrées : 1-9 -> Matrice 1
 *           10-18 -> Matrice 2
 *
 * note : les matrices sont entrées ligne par ligne
 *
 *      C  = A B
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   UQTR
 *   Copyright (c) mai 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME prod2mat3p3

#include "simstruc.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
```

```
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 18);/* number of inputs         */
    ssSetNumOutputs(       S, 9);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
  /*int lp, nOutputs;*/

  y[0]=u[0]*u[9]+u[1]*u[12]+u[2]*u[15];
  y[1]=u[0]*u[10]+u[1]*u[13]+u[2]*u[16];
  y[2]=u[0]*u[11]+u[1]*u[14]+u[2]*u[17];
  y[3]=u[3]*u[9]+u[4]*u[12]+u[5]*u[15];
  y[4]=u[3]*u[10]+u[4]*u[13]+u[5]*u[16];
  y[5]=u[3]*u[11]+u[4]*u[14]+u[5]*u[17];
  y[6]=u[6]*u[9]+u[7]*u[12]+u[8]*u[15];
  y[7]=u[6]*u[10]+u[7]*u[13]+u[8]*u[16];
  y[8]=u[6]*u[11]+u[7]*u[14]+u[8]*u[17];

}

/*
```

```
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Produit d'une matrice 3x3 par un vecteur 3x1 et addition du résultat à un vecteur 3x1 (Prodsom_matvect3)

```
/*
 * PRODMATVECTSOMVECT3P3 Produit d'une matrice par un vecteur et
addition du
 *                       résultat à un autre vecteur.
 *
 *      s  = A b + c
 *
 *   Entrées : 1-9 -> Matrice A
 *             10-12 -> vecteur b
 *             13-15 -> vecteur c
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   UQTR
 *   Copyright (c) mai 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME prodmatvectsomvect3p3
```

```
#include "simstruc.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 15);/* number of inputs         */
    ssSetNumOutputs(       S, 3);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
  /*int lp, nOutputs;*/

  y[0]=u[0]*u[9]+u[1]*u[10]+u[2]*u[11]+u[12];
  y[1]=u[3]*u[9]+u[4]*u[10]+u[5]*u[11]+u[13];
  y[2]=u[6]*u[9]+u[7]*u[10]+u[8]*u[11]+u[14];
```

```
}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Produit vectoriel (Prodvect)

```
/*
 * PRODVECT3 Produit vectoriel de 2 vecteurs
 *
 *      s = v1 X v2
 *
 *   Entrées : 1-3 -> vecteur 1
 *             4-6 -> vecteur 2
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   UQTR
 *   Copyright (c) mai 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME prodvect3
```

```
#include "simstruc.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 6);/* number of inputs         */
    ssSetNumOutputs(       S, 3);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
  /*int lp, nOutputs;*/

  y[0]=u[1]*u[5]-u[2]*u[4];
  y[1]=-u[0]*u[5]+u[2]*u[3];
  y[2]=u[0]*u[4]-u[1]*u[3];
```

```
}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Produit d'une matrice 3x3 par un vecteur 3x1 (Prod_matvect3)

```
/*
 * PRODMATVECT3P3 Produit d'une matrice par un vecteur.
 *
 *      Vs  = M V
 *
 *   Entrées : 1-9 -> Matrice
 *             10-12 -> Vecteur
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   UQTR
 *   Copyright (c) mai 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME prodmatvect3p3
```

```
#include "simstruc.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 12);/* number of inputs         */
    ssSetNumOutputs(       S, 3);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
  /*int lp, nOutputs;*/

  y[0]=u[0]*u[9]+u[1]*u[10]+u[2]*u[11];
  y[1]=u[3]*u[9]+u[4]*u[10]+u[5]*u[11];
  y[2]=u[6]*u[9]+u[7]*u[10]+u[8]*u[11];
```

```
}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Transposée d'une matrice 3x3 (Transposée)

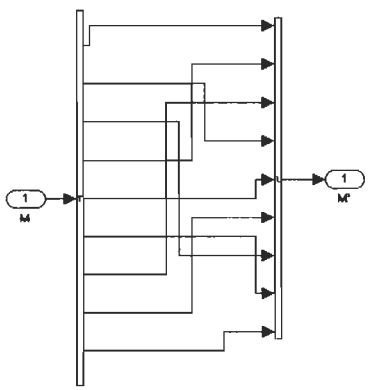

Figure C. 2 : Transposée d'une matrice 3x3

Sélectionneur d'un élément d'un vecteur de dimension quelconque (Choixvect)

```
/*
 *    CHOIXVECT Démultiplexeur dynamique d'un scalaire dans un vecteur
 *
 *    Entree = [#articulation; vecteur]
 *
 *      Sortie = [élément scalaire pointé]
 *
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   Copyright (c) mai 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME choixvect

#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, DYNAMICALLY_SIZED);/* number of inputs
*/
    ssSetNumOutputs(       S, 1);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
```

```
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
   int      i;

   i=u[0];
   y[0]=u[i];

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Sélectionneur d'un élément d'un vecteur de dimension 3 (Choixvect3)

```
/*
 *    CHOIXVECT3 Démultiplexeur dynamique d'un vecteur de 3 éléments
 *               dans un vecteur principal.
 *
 *    Entree = [#articulation; vecteur]
 *
 *      Sortie = [vecteur pointé]
 *
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   Copyright (c) mai 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME choixvect3

#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, DYNAMICALLY_SIZED);/* number of inputs
*/
    ssSetNumOutputs(       S, 3);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
```

```
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
   int      i;

   i=u[0]-1;
   y[0]=u[i*3+1];
   y[1]=u[i*3+2];
   y[2]=u[i*3+3];

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
```

```
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Sélectionneur d'un élément d'un vecteur de dimension 9 (Choixvect9)

```
/*
 *    CHOIXVECT9 Démultiplexeur dynamique d'un vecteur de 9 éléments
 *               dans un vecteur principal.
 *
 *    Entree = [#articulation; vecteur]
 *
 *      Sortie = [vecteur pointé]
 *
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   Copyright (c) mai 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME choixvect9

#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, DYNAMICALLY_SIZED);/* number of inputs
*/
    ssSetNumOutputs(       S, 9);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}
```

```
/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
   int      i;

   i=u[0]-1;
   y[0]=u[i*9+1];
   y[1]=u[i*9+2];
   y[2]=u[i*9+3];
   y[3]=u[i*9+4];
   y[4]=u[i*9+5];
   y[5]=u[i*9+6];
   y[6]=u[i*9+7];
   y[7]=u[i*9+8];
   y[8]=u[i*9+9];

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
```

```
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Projection d'un vecteur sur l'axe z (Xzidi)

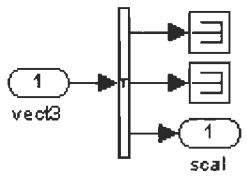

Figure C. 3 : Projection d'un vecteur de dimension 3 sur l'axe z

Résolution d'équation (pour y = Ax, trouve x) utilisant la décomposition LU et pour une matrice A symétrique (divmat)

```
/*
 * DIVMAT : Résolution d'un système d'équation linaire par la méthode de
 *          décomposition LU.
 *      Soit :
 *
 *                      Ax=b
 *
 *          Nous voulons trouver :
 *
 *      x=b/A
 *
 *        où b est le vecteur de solutions
 *                A est la matrice
 *                x est le vecteur à résoudre
 *
 *   Par : Brian Moore
 *         Martin de Montigny
 *
 *   Opal-rt
 *   Copyright (c) mai 1998
```

```
 *   Tous droits réservés
 *   version 1.1
 */

#define S_FUNCTION_NAME divmat

#include "simstruc.h"
#include "libconst.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, NIN);/* number of inputs         */
    ssSetNumOutputs(       S, NOUT);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */
```

```
static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
  /*int lp, nOutputs;*/

int i,j,k,n,n2,nt;
double sum, ll[NOUT2], z[NOUT], b[NOUT];

/* Production de la matrice temporaire */
for(i=0;i<NOUT2;i++)
{
   ll[i]=u[i];
}
for(i=0; i<NOUT; i++)
{
  b[i] = u[NOUT2+i];
}

/*  Decomposition LU   */

  for(k=0; k<NOUT;k++)
  {
    for(i=0; i<k; i++)
    {
       sum = 0.0;
       for(j=0; j<i; j++)
       {
         sum += ll[NOUT*i + j]*ll[NOUT*k + j];
       }
       ll[k*NOUT+i] = (ll[k*NOUT+i] - sum)/ll[i*NOUT+i];

    }
    sum = 0.0;
    for(j=0; j<k; j++)
    {
     sum += ll[NOUT*k+j]*ll[NOUT*k+j];
    }
    ll[NOUT*k+k] = sqrt(ll[NOUT*k+k] - sum);

  }

  /*  Calcul de z   :  Lz = b   */
  for(i=0; i<NOUT; i++)
  {
    sum = b[i];
    for(j=0; j<i; j++)
    {
      sum = sum - z[j]*ll[i*NOUT+j];
    }
    z[i] = sum/ll[i*NOUT+i];
  }

 /*  Calcul de y   :  Uy = z   */

  for(i=NOUT-1; i>=0; i--)
  {
    sum = z[i];
```

```
    for(j=NOUT-1; j>i; j--)
    {
      sum = sum - y[j]*ll[j*NOUT+i];
    }
    y[i] = sum/ll[i*NOUT+i];
  }
}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file?
*/
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Blocs non compilés (seulement des blocs Simulink) de calcul de la dynamique inverse utilisant la formulation de Newton-Euler

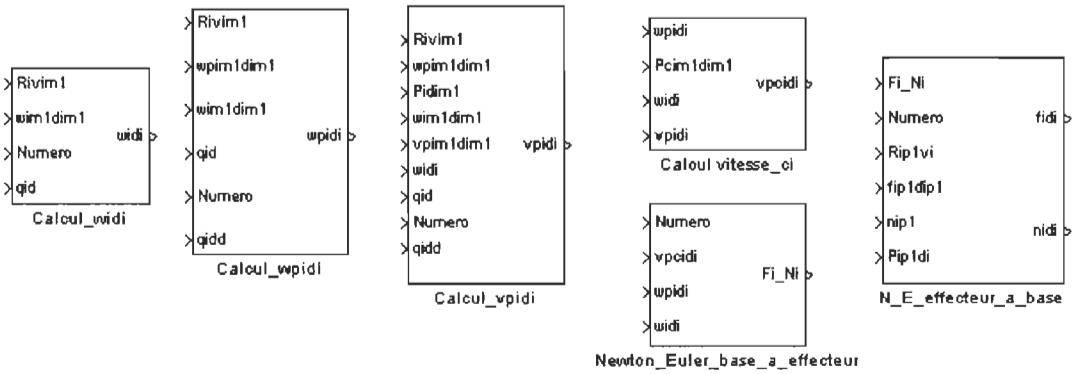

Figure C. 4 : Librairie des blocs de calcul de la dynamique inverse sous la formulation de Newton-Euler en utilisant seulement des blocs Simulink

Calcul de la vitesse rotative d'un menbre par rapport à la base et représentée dans le référentiel du membre (Calcul_widi)

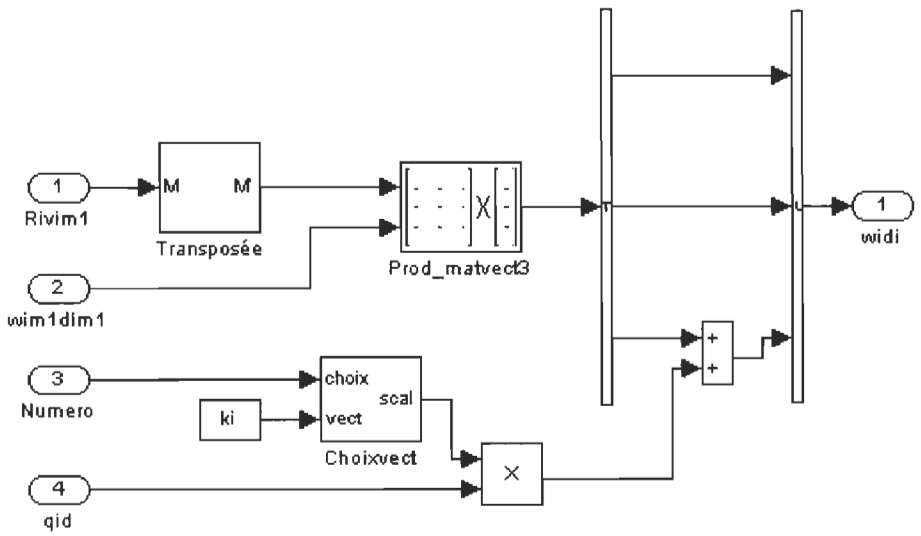

Figure C. 5 : Calcul de la vitesse rotative d'un membre par rapport à la base et représentée dans le référentiel du membre (avec seulement des blocs Simulink)

Calcul de l'accélération rotative d'un menbre par rapport à la base et représentée dans le référentiel du membre (Calcul_wpidi)

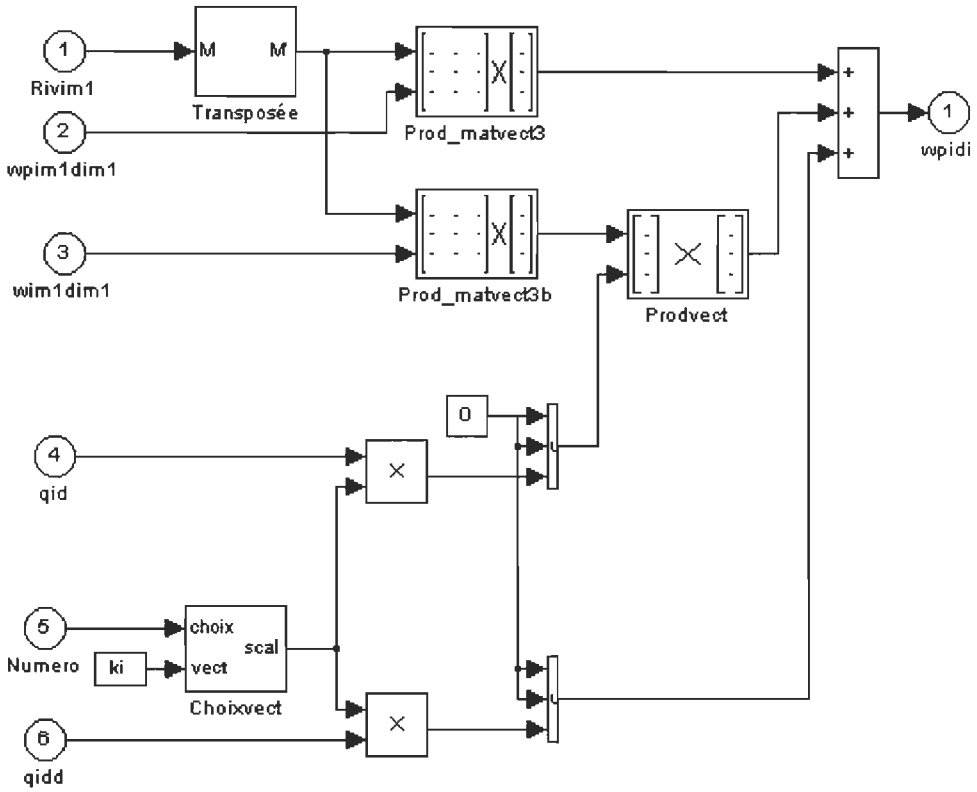

Figure C. 6 : Calcul de l'accélération rotative d'un menbre par rapport à la base et représentée dans le référentiel du membre (avec seulement des blocs Simulink)

Calcul de l'accélération linéaire d'un menbre par rapport à la base et représentée dans le référentiel du membre (Calcul_vpidi)

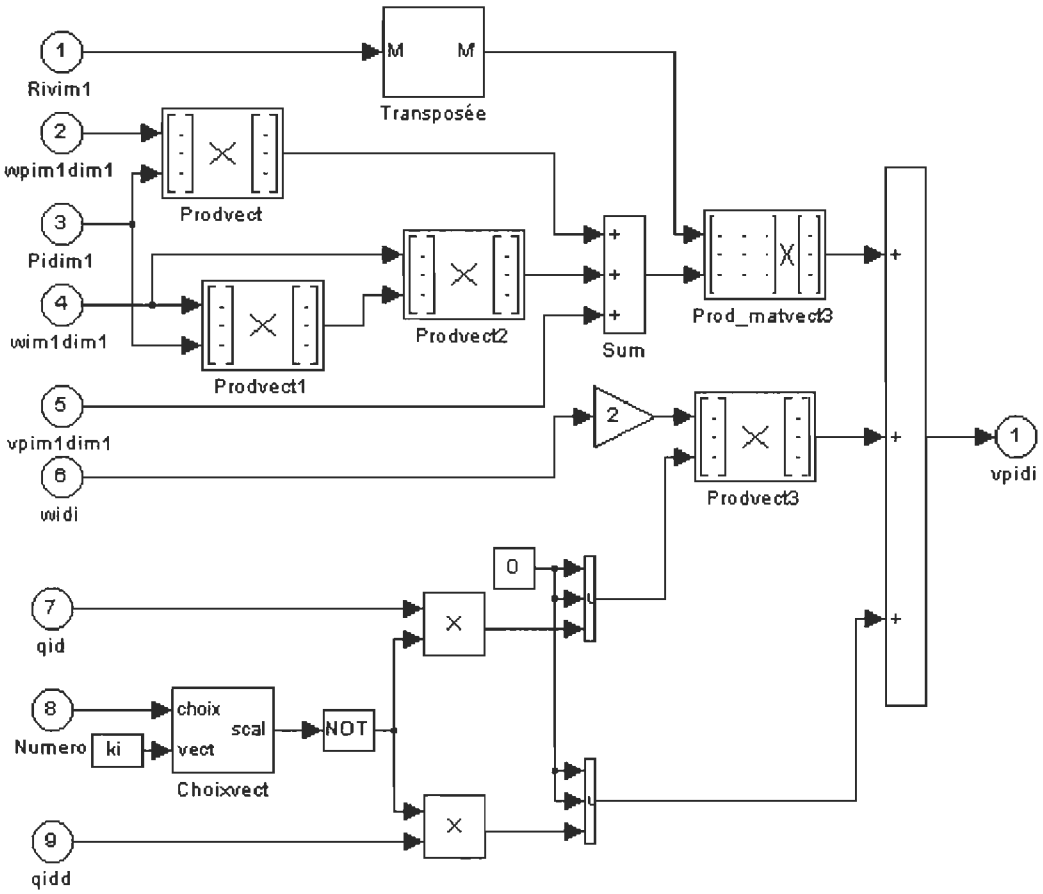

Figure C. 7 : Calcul de l'accélération linéaire d'un menbre par rapport à la base et représentée dans le référentiel du membre (avec seulement des blocs Simulink)

Calcul de l'accélération linéaire du centre de masse d'un membre par rapport à la base et représentée dans le référentiel du membre (Calcul vitesse_ci)

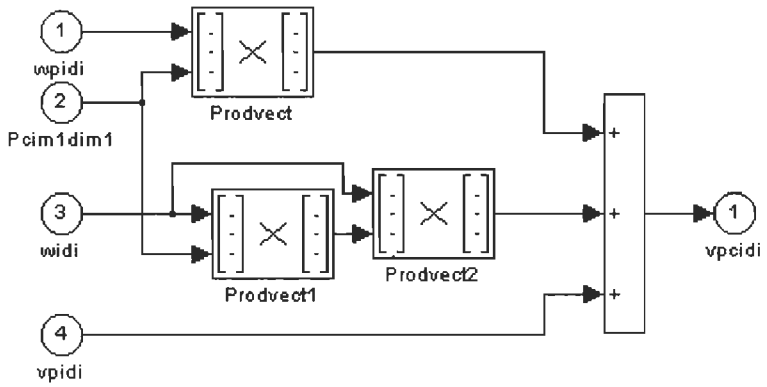

Figure C. 8 : Calcul de l'accélération linéaire du centre de masse d'un membre par rapport à la base et représentée dans le référentiel du membre (avec seulement des blocs Simulink)

Calcul des force et couple d'inertie agissant sur un membre (Newton_Euler_base_a_effecteur)

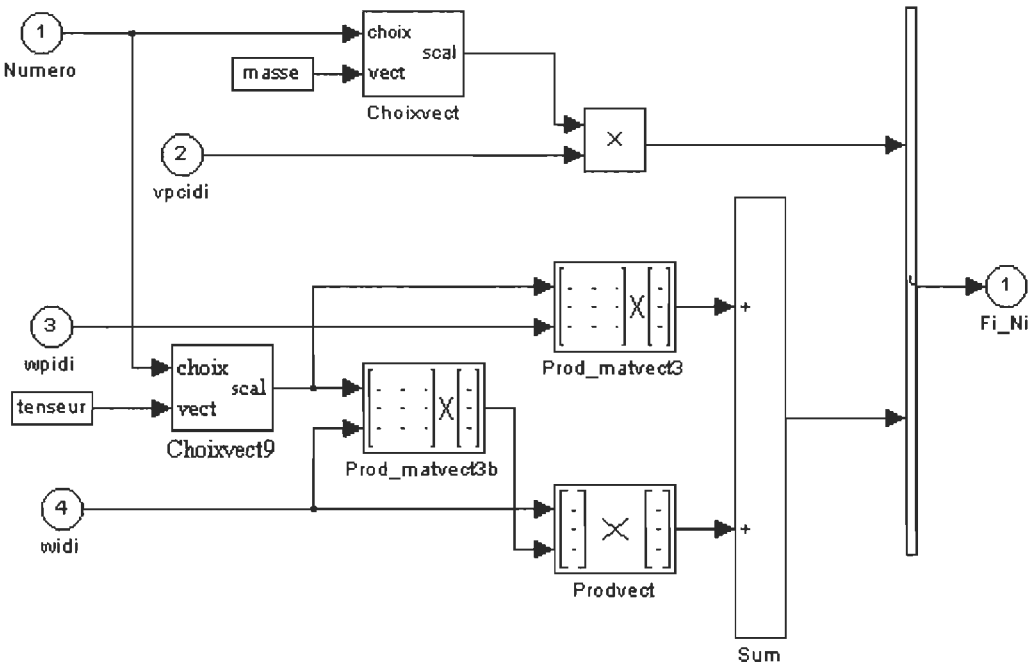

Figure C. 9 : Calcul des forces et couples agissant sur un membre avec seulement des blocs Simulink (itération de la base vers l'effecteur)

Calcul des forces/couples articulaires résultant des forces/couples agissant sur sur chaque lien (itération de l'effecteur vers la base) (N_E_effecteur_a_base)

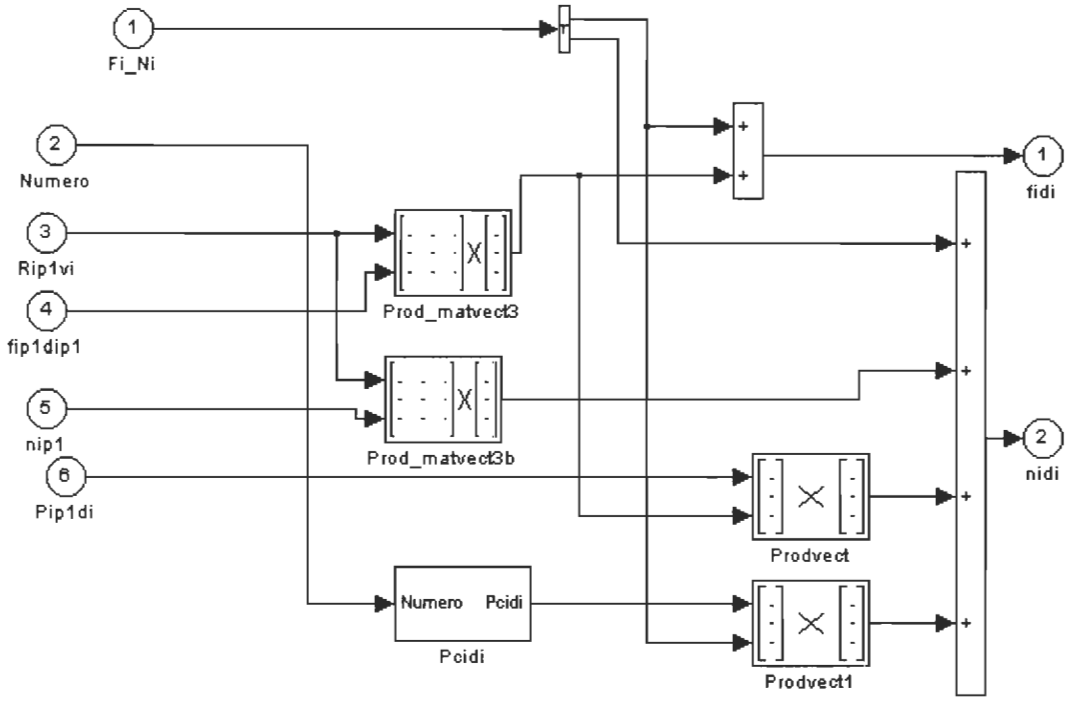

Figure C. 10 : Calcul des forces/couples articulaires d'un membre avec seulement des blocs Simulink (itération de l'effecteur à la base)

Blocs compilés de calcul de la dynamique inverse sous la forme de Newton-Euler (chaque fonction est une équation de calcul de vitesse, accélération ou de forces/couples)

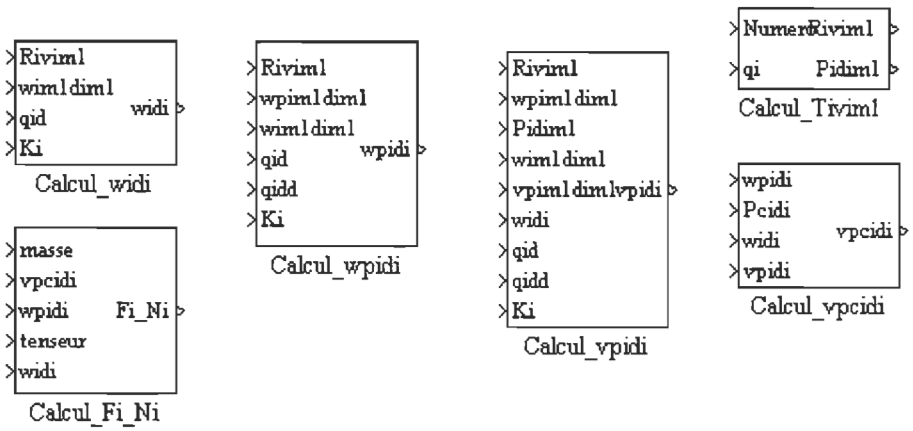

Figure C. 11 Librairie des blocs de calcul de la dynamique inverse équation par équation avec la formulation de Newton-Euler

Fonction compilée de calcul de la vitesse rotative d'un membre par rapport à la base et représentée dans le référentiel du membre (Calcul_widi)

```
/*
 * CALCULWIDI Calcul de la vitesse rotative des référentiels.
 *
 *
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   Copyright (c) juin 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME calculwidi

#include "simstruc.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
```

```
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 14);/* number of inputs         */
    ssSetNumOutputs(       S, 3);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

  y[0]=u[0]*u[9]+u[3]*u[10]+u[6]*u[11];
  y[1]=u[1]*u[9]+u[4]*u[10]+u[7]*u[11];
  y[2]=u[2]*u[9]+u[5]*u[10]+u[8]*u[11]+u[13]*u[12];

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
```

```
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Fonction compilée de calcul de l'accélération rotative d'un membre par rapport à la base et représentée dans le référentiel du membre (Calcul_wpidi)

```
/*
 * CALCULWPIDI Calcul de l'accélération rotative des référentiels.
 *
 *
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   Copyright (c) juin 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME calculwpidi

#include "simstruc.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
```

```
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 18);/* number of inputs         */
    ssSetNumOutputs(       S, 3);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{


y[0]=u[0]*u[9]+u[3]*u[10]+u[6]*u[11]+u[17]*(u[1]*u[12]+u[4]*u[13]+u[7]*u
[14])*u[15];
  y[1]=u[1]*u[9]+u[4]*u[10]+u[7]*u[11]-
u[17]*(u[0]*u[12]+u[3]*u[13]+u[6]*u[14])*u[15];
  y[2]=u[2]*u[9]+u[5]*u[10]+u[8]*u[11]+u[17]*u[16];

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
```

```
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Fonction compilée de calcul de l'accélération linéaire d'un membre par rapport à la base et représentée dans le référentiel du membre (Calcul_vpidi)

```
/*
 * CALCULVPIDI Calcul de l'accélération linéaire des référentiels.
 *
 *
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   Copyright (c) mai 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME calculvpidi

#include "simstruc.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
```

```
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 27);/* number of inputs         */
    ssSetNumOutputs(       S, 3);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
  double g1, g2, g3, g4, g5, g6, g7, g8, g9, g4t, g6t, g8t, g10, g11,
g12, g13;


  /* Calcul des groupes répétitifs */
  g1=u[10]*u[14]-u[11]*u[13];
  g2=u[11]*u[12]-u[9]*u[14];
  g3=u[9]*u[13]-u[10]*u[12];
  g4t=u[15]*u[13]-u[16]*u[12];
  g4=g4t*u[16];
  g5=g4t*u[15]-u[19];
  g6t=u[17]*u[12]-u[15]*u[14];
  g6=g6t*u[15];
```

```
  g7=g6t*u[17]-u[18];
  g8t=u[16]*u[14]-u[17]*u[13];
  g8=g8t*u[17];
  g9=g8t*u[16]-u[20];
  g10=1-u[26];

  g11=g1+g4-g7;
  g12=g2+g8-g5;
  g13=g3+g6-g9;

  y[0]=u[0]*(g11)+u[3]*(g12)+u[6]*(g13)+2*g10*u[22]*u[24];
  y[1]=u[1]*(g11)+u[4]*(g12)+u[7]*(g13)-2*g10*u[21]*u[24];
  y[2]=u[2]*(g11)+u[5]*(g12)+u[8]*(g13)+g10*u[25];

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Fonction compilée de calcul de l'accélération linéaire du centre de masse d'un membre par rapport à la base et représentée dans le référentiel du membre (Calcul_vpcidi)

```
/*
 * CALCULVPCIDI Calcul de l'accélération linéaire des masses.
 *
 *
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   Copyright (c) juin 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME calculvpcidi

#include "simstruc.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 12);/* number of inputs         */
    ssSetNumOutputs(       S, 3);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
```

```
/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
  double g1, g2, g3;

  g1=u[6]*u[4]-u[7]*u[3];
  g2=u[8]*u[3]-u[6]*u[5];
  g3=u[7]*u[5]-u[8]*u[4];

  y[0]=u[1]*u[5]-u[2]*u[4]+u[7]*g1-u[8]*g2+u[9];
  y[1]=u[2]*u[3]-u[0]*u[5]+u[8]*g3-u[6]*g1+u[10];
  y[2]=u[0]*u[4]-u[1]*u[3]+u[6]*g2-u[7]*g3+u[11];

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Fonction compilée de calcul de la cinématique directe d'un membre (Calcul_Tivim1)

```
/*
 *    TIVIM1 Calcul de la matrice de transformation Tivim1
 *                (Matrice Rivim1 et vecteur Pidim1)
 *
 *    Entree = [Paramètres DH mod; qi]
 *
 *    Les paramètres DH mod sont :  Numero, ki, aim1, alphaim1, di et
thetai.
 *
 *  ki désigne le type d'articulation(0:prismatique, 1:rotative)
 *
 *      Sortie = [Rivim1; Pidim1]
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   UQTR
 *   Copyright (c) mai 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME tivim1

#include "simstruc.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, DYNAMICALLY_SIZED);    /* number of
inputs  */
    ssSetNumOutputs(       S, 12);  /* number of outputs
*/
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
```

```
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

   /* sin(thetai), cos(thetai), sin(alphaim1), cos(alphaim1), di */
   double sthetai,cthetai,salphaim1,calphaim1,di;
   int nentrees,posaim1,posalphaim1,posdi,posthetai,posqi,nart;

   /* Calcul du nombre d'articulations */
   nart=(ssGetNumInputs(S)-1)/6;

   /* Calculs des positions des paramètres DHmod */
   posaim1=nart+((int) u[0]);
   posalphaim1=2*nart+((int) u[0]);
   posdi=3*nart+((int) u[0]);
   posthetai=4*nart+((int) u[0]);
   posqi=5*nart+((int) u[0]);

   /* si ki=0 */
   if (u[((int) u[0])]==0)
   {
      sthetai=sin(u[posthetai]);
      cthetai=cos(u[posthetai]);
      di=u[posqi]+u[posdi];
   }
   else
   {
      sthetai=sin(u[posqi]+u[posthetai]);
      cthetai=cos(u[posqi]+u[posthetai]);
      di=u[posdi];
   }

   salphaim1=sin(u[posalphaim1]);
   calphaim1=cos(u[posalphaim1]);
```

```
    /* Production de la matrice de rotation Rivim1 */
    y[0] = cthetai;
    y[1] = -sthetai;
    y[2] = 0;
    y[3] = sthetai*calphaim1;
    y[4] = cthetai*calphaim1;
    y[5] = -salphaim1;
    y[6] = sthetai*salphaim1;
    y[7] = cthetai*salphaim1;
    y[8] = calphaim1;

    /* Production du vecteur de translation Pidim1 */
    y[9] = u[posaim1];
    y[10] = -salphaim1*di;
    y[11] = calphaim1*di;

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Fonction compilée de calcul des forces/couples d'inertie agissant sur un membre (Calcul_Fi_Ni)

```
/*
 * CALCULFINI Calcul des forces et couples d'inertie des membres du
manipulateur.
 *
 *
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   Copyright (c) juin 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME calculFiNi

#include "simstruc.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 19);/* number of inputs         */
    ssSetNumOutputs(       S, 6);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
```

```
/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

  y[0]=u[0]*u[1];
  y[1]=u[0]*u[2];
  y[2]=u[0]*u[3];
  y[3]=u[7]*u[4]+u[17]*u[18]*(u[15]-u[11]);
  y[4]=u[11]*u[5]+u[16]*u[18]*(u[7]-u[15]);
  y[5]=u[15]*u[6]+u[16]*u[17]*(u[11]-u[7]);

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Librairie des blocs d'appel des fonctions de calcul de l'algorithme de commande de position de Slotine et Li

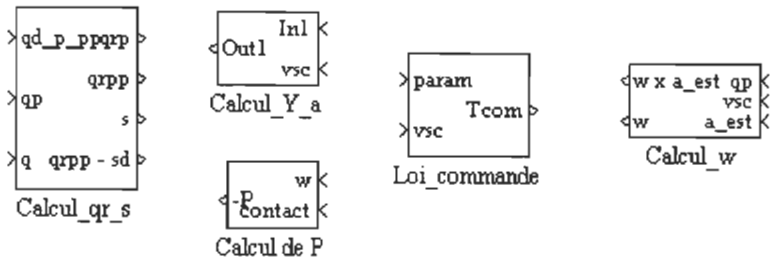

Figure C. 12 : Librairie des fonctions de commande adaptative de position par l'algorithme de Slotine et Li

Calculs généraux (erreur de suivi, erreur de position, vitesse et accélération, etc.) (Calcul_qr_s)

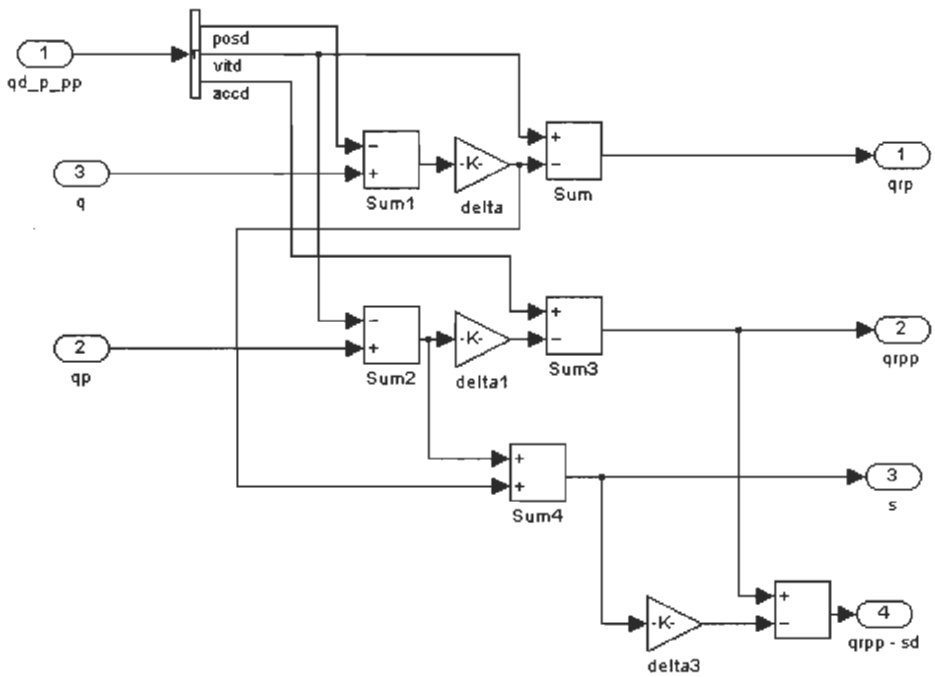

Figure C. 13 : Calculs généraux nécessaires pour l'algorithme de commande adaptatif de position de Slotine et Li

Le vecteur d'entrée qd_p_pp contient les positions, vitesses et accélérations désirées. Les entrées q et qd sont les position et vitesse réelles. Les sorties qrp et qrpp sont les versions

modifiées de la vitesse et de l'accélération. La sortie s est l'erreur de suivi et la sortie « qrpp - sd » est l'accélération fournie à la loi de commande de façon à inclure implicitement un contrôleur PD.

Calcul de la dérivée des paramètres estimés (Calcul_Y_a)

```
/*
 * calcul_y_a : Calcul de Y et a (partiel) (pour calculer la derivée de
a)
 *
 *   Entrées : qp (vitesse articulaire)
 *             e (erreur de couple causée par l'erreur d'observation des
paramètres)
 *             s (erreur de poursuite)
 *             qrp (vitesse articulaire modifiée)
 *             qrpp (accélération articulaire modifiée)
 *             matrice W
 *             matrice P
 *             matrice R (constante)
 *             vecteur de précalcul de sinus et de cosinus de q
(position articulaire)
 *
 *   Sortie : 1-3 -> a (paramètres estimés : masses)
 *
 *   Par : Martin de Montigny
 *   UQTR : Laboratoire de commande
 *   Copyright (c) 1998-99
 *   Tous droits réservés
 *   version 2.0
 */

#define S_FUNCTION_NAME calcul_y_a
#define l2L2 0.24
#define L2L2 0.64
#define l0l0 0.16
#define L1L2 0.64
#define l2L1 0.24
#define l1l1 0.16
#define l2l2 0.09
#define L1L1 0.64
#define l1L1 0.32
#define l1 0.4
#define L2 0.8
#define l0 0.4
#define l2 0.3
#define L1 0.8
#define L3 0.6
#define L4 0.1
```

```
#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(S,51);
    ssSetNumOutputs(S,3);
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
```

```
y[0]=u[24]*((l0l0*u[12]+9.81000000000*u[46]*l0)*u[6]+(u[15]*u[33]+u[18]*
u[36]+u[21]*u[39])*u[3]+(u[15]*u[34]+u[18]*u[37]+u[21]*u[40])*u[4]+(u[15
]*u[35]+u[18]*u[38]+u[21]*u[41])*u[5])+u[25]*((u[12]*l1l1+u[12]*L1L1+2.*
u[12]*u[49]*l1L1+u[13]*u[49]*l1L1+l1l1*u[13]-1.*u[1]*u[48]*l1L1*u[9]-
1.*u[10]*u[0]*u[48]*l1L1-
1.*u[10]*u[1]*u[48]*l1L1+9.81000000000*u[43]*l1+9.81000000000*u[46]*L1)*
u[6]+(u[12]*u[49]*l1L1+u[12]*l1l1+l1l1*u[13]+u[0]*u[48]*l1L1*u[9]+9.8100
0000000*u[43]*l1)*u[7]+(u[16]*u[33]+u[19]*u[36]+u[22]*u[39])*u[3]+(u[16]
*u[34]+u[19]*u[37]+u[22]*u[40])*u[4]+(u[16]*u[35]+u[19]*u[38]+u[22]*u[41
])*u[5])+u[26]*((-1.*u[10]*u[0]*u[44]*l2L1-1.*u[10]*u[0]*u[48]*L1L2-
1.*u[9]*u[2]*u[44]*l2L1-1.*u[9]*u[47]*u[2]*l2L2-1.*u[9]*u[1]*u[44]*l2L1-
1.*u[9]*u[1]*u[48]*L1L2+u[13]*u[45]*l2L1+u[13]*u[49]*L1L2+2.*u[13]*u[50]
*l2L2+2.*u[12]*u[45]*l2L1+2.*u[12]*u[49]*L1L2+2.*u[12]*u[50]*l2L2+u[12]*
L1L1-1.*u[10]*u[1]*u[48]*L1L2-1.*u[11]*u[47]*u[0]*l2L2-
1.*u[10]*u[2]*u[44]*l2L1-
1.*u[10]*u[1]*u[44]*l2L1+u[14]*u[45]*l2L1+u[14]*u[50]*l2L2-
1.*u[11]*u[47]*u[2]*l2L2-1.*u[11]*u[1]*u[44]*l2L1-
1.*u[11]*u[47]*u[1]*l2L2-1.*u[11]*u[0]*u[44]*l2L1-
1.*u[11]*u[2]*u[44]*l2L1+9.81000000000*u[42]*l2+9.81000000000*u[43]*L2+u
[12]*L2L2+u[13]*l2l2+u[13]*L2L2+u[12]*l2l2+9.81000000000*u[46]*L1+l2l2*u
[14]-
1.*u[47]*u[2]*l2L2*u[10])*u[6]+(2.*u[12]*u[50]*l2L2+u[12]*L2L2+u[12]*u[4
9]*L1L2+u[12]*l2l2+u[12]*u[45]*l2L1+2.*u[13]*u[50]*l2L2+u[13]*L2L2+u[13]
*l2l2+u[14]*u[50]*l2L2+l2l2*u[14]+u[9]*u[0]*u[48]*L1L2+u[9]*u[0]*u[44]*l
2L1-1.*u[9]*u[47]*u[2]*l2L2-1.*u[47]*u[2]*l2L2*u[10]-
1.*u[11]*u[47]*u[0]*l2L2-1.*u[11]*u[47]*u[1]*l2L2-
1.*u[11]*u[47]*u[2]*l2L2+9.81000000000*u[43]*L2+9.81000000000*u[42]*l2)*
u[7]+(u[12]*u[50]*l2L2+u[12]*l2l2+u[12]*u[45]*l2L1+u[13]*u[50]*l2L2+u[13
]*l2l2+l2l2*u[14]+u[9]*u[47]*u[0]*l2L2+u[9]*u[0]*u[44]*l2L1+u[9]*u[47]*u
[1]*l2L2+u[10]*u[47]*u[0]*l2L2+u[10]*u[47]*u[1]*l2L2+9.81000000000*u[42]
*l2)*u[8]+(u[17]*u[33]+u[20]*u[36]+u[23]*u[39])*u[3]+(u[17]*u[34]+u[20]*
u[37]+u[23]*u[40])*u[4]+(u[17]*u[35]+u[20]*u[38]+u[23]*u[41])*u[5]);
y[1]=u[27]*((l0l0*u[12]+9.81000000000*u[46]*l0)*u[6]+(u[15]*u[33]+u[18]*
u[36]+u[21]*u[39])*u[3]+(u[15]*u[34]+u[18]*u[37]+u[21]*u[40])*u[4]+(u[15
]*u[35]+u[18]*u[38]+u[21]*u[41])*u[5])+u[28]*((u[12]*l1l1+u[12]*L1L1+2.*
u[12]*u[49]*l1L1+u[13]*u[49]*l1L1+l1l1*u[13]-1.*u[1]*u[48]*l1L1*u[9]-
1.*u[10]*u[0]*u[48]*l1L1-
1.*u[10]*u[1]*u[48]*l1L1+9.81000000000*u[43]*l1+9.81000000000*u[46]*L1)*
u[6]+(u[12]*u[49]*l1L1+u[12]*l1l1+l1l1*u[13]+u[0]*u[48]*l1L1*u[9]+9.8100
0000000*u[43]*l1)*u[7]+(u[16]*u[33]+u[19]*u[36]+u[22]*u[39])*u[3]+(u[16]
*u[34]+u[19]*u[37]+u[22]*u[40])*u[4]+(u[16]*u[35]+u[19]*u[38]+u[22]*u[41
])*u[5])+u[29]*((-1.*u[10]*u[0]*u[44]*l2L1-1.*u[10]*u[0]*u[48]*L1L2-
1.*u[9]*u[2]*u[44]*l2L1-1.*u[9]*u[47]*u[2]*l2L2-1.*u[9]*u[1]*u[44]*l2L1-
1.*u[9]*u[1]*u[48]*L1L2+u[13]*u[45]*l2L1+u[13]*u[49]*L1L2+2.*u[13]*u[50]
*l2L2+2.*u[12]*u[45]*l2L1+2.*u[12]*u[49]*L1L2+2.*u[12]*u[50]*l2L2+u[12]*
L1L1-1.*u[10]*u[1]*u[48]*L1L2-1.*u[11]*u[47]*u[0]*l2L2-
1.*u[10]*u[2]*u[44]*l2L1-
1.*u[10]*u[1]*u[44]*l2L1+u[14]*u[45]*l2L1+u[14]*u[50]*l2L2-
1.*u[11]*u[47]*u[2]*l2L2-1.*u[11]*u[1]*u[44]*l2L1-
1.*u[11]*u[47]*u[1]*l2L2-1.*u[11]*u[0]*u[44]*l2L1-
1.*u[11]*u[2]*u[44]*l2L1+9.81000000000*u[42]*l2+9.81000000000*u[43]*L2+u
[12]*L2L2+u[13]*l2l2+u[13]*L2L2+u[12]*l2l2+9.81000000000*u[46]*L1+l2l2*u
[14]-
1.*u[47]*u[2]*l2L2*u[10])*u[6]+(2.*u[12]*u[50]*l2L2+u[12]*L2L2+u[12]*u[4
9]*L1L2+u[12]*l2l2+u[12]*u[45]*l2L1+2.*u[13]*u[50]*l2L2+u[13]*L2L2+u[13]
*l2l2+u[14]*u[50]*l2L2+l2l2*u[14]+u[9]*u[0]*u[48]*L1L2+u[9]*u[0]*u[44]*l
2L1-1.*u[9]*u[47]*u[2]*l2L2-1.*u[47]*u[2]*l2L2*u[10]-
```

```
1.*u[11]*u[47]*u[0]*l2L2-1.*u[11]*u[47]*u[1]*l2L2-
1.*u[11]*u[47]*u[2]*l2L2+9.81000000000*u[43]*L2+9.81000000000*u[42]*l2)*
u[7]+(u[12]*u[50]*l2L2+u[12]*l2l2+u[12]*u[45]*l2L1+u[13]*u[50]*l2L2+u[13
]*l2l2+l2l2*u[14]+u[9]*u[47]*u[0]*l2L2+u[9]*u[0]*u[44]*l2L1+u[9]*u[47]*u
[1]*l2L2+u[10]*u[47]*u[0]*l2L2+u[10]*u[47]*u[1]*l2L2+9.81000000000*u[42]
*l2)*u[8]+(u[17]*u[33]+u[20]*u[36]+u[23]*u[39])*u[3]+(u[17]*u[34]+u[20]*
u[37]+u[23]*u[40])*u[4]+(u[17]*u[35]+u[20]*u[38]+u[23]*u[41])*u[5]);
y[2]=u[30]*((l0l0*u[12]+9.81000000000*u[46]*l0)*u[6]+(u[15]*u[33]+u[18]*
u[36]+u[21]*u[39])*u[3]+(u[15]*u[34]+u[18]*u[37]+u[21]*u[40])*u[4]+(u[15
]*u[35]+u[18]*u[38]+u[21]*u[41])*u[5])+u[31]*((u[12]*l1l1+u[12]*L1L1+2.*
u[12]*u[49]*l1L1+u[13]*u[49]*l1L1+l1l1*u[13]-1.*u[1]*u[48]*l1L1*u[9]-
1.*u[10]*u[0]*u[48]*l1L1-
1.*u[l0]*u[1]*u[48]*l1L1+9.81000000000*u[43]*l1+9.81000000000*u[46]*L1)*
u[6]+(u[12]*u[49]*l1L1+u[12]*l1l1+l1l1*u[13]+u[0]*u[48]*l1L1*u[9]+9.8100
0000000*u[43]*l1)*u[7]+(u[16]*u[33]+u[19]*u[36]+u[22]*u[39])*u[3]+(u[16]
*u[34]+u[19]*u[37]+u[22]*u[40])*u[4]+(u[16]*u[35]+u[19]*u[38]+u[22]*u[41
])*u[5])+u[32]*((-1.*u[10]*u[0]*u[44]*l2L1-1.*u[10]*u[0]*u[48]*L1L2-
1.*u[9]*u[2]*u[44]*l2L1-1.*u[9]*u[47]*u[2]*l2L2-1.*u[9]*u[1]*u[44]*l2L1-
1.*u[9]*u[1]*u[48]*L1L2+u[13]*u[45]*l2L1+u[13]*u[49]*L1L2+2.*u[13]*u[50]
*l2L2+2.*u[12]*u[45]*l2L1+2.*u[12]*u[49]*L1L2+2.*u[12]*u[50]*l2L2+u[12]*
L1L1-1.*u[10]*u[1]*u[48]*L1L2-1.*u[11]*u[47]*u[0]*l2L2-
1.*u[10]*u[2]*u[44]*l2L1-
1.*u[10]*u[1]*u[44]*l2L1+u[14]*u[45]*l2L1+u[14]*u[50]*l2L2-
1.*u[11]*u[47]*u[2]*l2L2-1.*u[11]*u[1]*u[44]*l2L1-
1.*u[11]*u[47]*u[1]*l2L2-1.*u[11]*u[0]*u[44]*l2L1-
1.*u[11]*u[2]*u[44]*l2L1+9.81000000000*u[42]*l2+9.81000000000*u[43]*L2+u
[12]*L2L2+u[13]*l2l2+u[13]*L2L2+u[12]*l2l2+9.81000000000*u[46]*L1+l2l2*u
[14]-
1.*u[47]*u[2]*l2L2*u[10])*u[6]+(2.*u[12]*u[50]*l2L2+u[12]*L2L2+u[12]*u[4
9]*L1L2+u[12]*l2l2+u[12]*u[45]*l2L1+2.*u[13]*u[50]*l2L2+u[13]*L2L2+u[13]
*l2l2+u[14]*u[50]*l2L2+l2l2*u[14]+u[9]*u[0]*u[48]*L1L2+u[9]*u[0]*u[44]*l
2L1-1.*u[9]*u[47]*u[2]*l2L2-1.*u[47]*u[2]*l2L2*u[10]-
1.*u[11]*u[47]*u[0]*l2L2-1.*u[11]*u[47]*u[1]*l2L2-
1.*u[11]*u[47]*u[2]*l2L2+9.81000000000*u[43]*L2+9.81000000000*u[42]*l2)*
u[7]+(u[12]*u[50]*l2L2+u[12]*l2l2+u[12]*u[45]*l2L1+u[13]*u[50]*l2L2+u[13
]*l2l2+l2l2*u[14]+u[9]*u[47]*u[0]*l2L2+u[9]*u[0]*u[44]*l2L1+u[9]*u[47]*u
[1]*l2L2+u[l0]*u[47]*u[0]*l2L2+u[10]*u[47]*u[1]*l2L2+9.81000000000*u[42]
*l2)*u[8]+(u[17]*u[33]+u[20]*u[36]+u[23]*u[39])*u[3]+(u[17]*u[34]+u[20]*
u[37]+u[23]*u[40])*u[4]+(u[17]*u[35]+u[20]*u[38]+u[23]*u[41])*u[5]);


}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
```

```
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Calcul de l'opposé de la matrice inverse de covariance -P (Calcul de P)

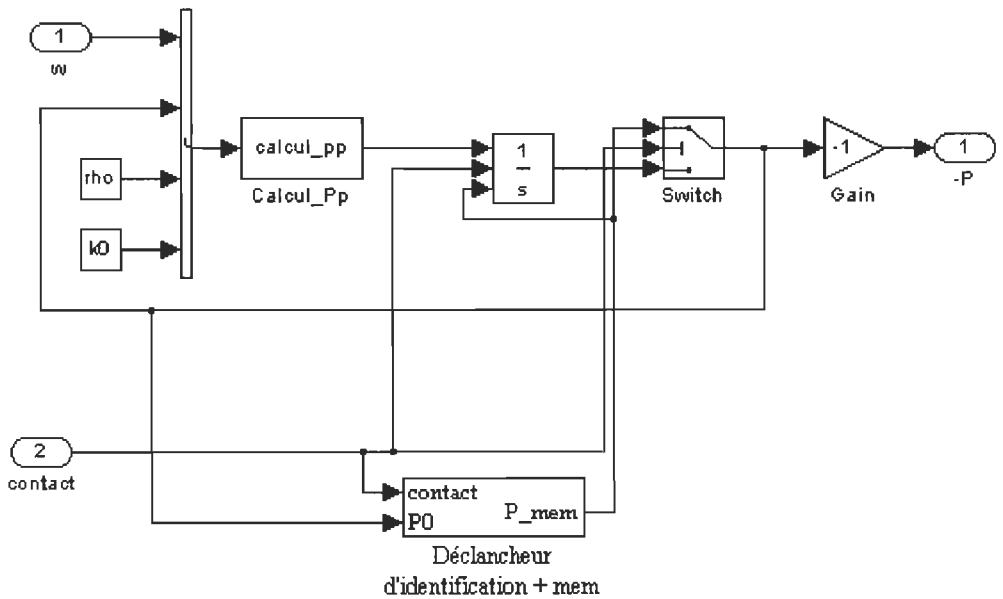

Figure C. 14 : Calcul de l'opposé de la matrice inverse de covariance -P

```
/*
 * CALCUL_Pp : Calcul de P dérivé (pour calculer la derivée de a)
 *
 *   Entrées : matrice W
 *             matrice P
 *             rho (facteur d'oubli)
 *             k0 (limite de module de la matrice P)
 *
```

```
 *   Sortie : matrice P dérivée
 *
 *
 *   Par : Martin de Montigny
 *   UQTR : Laboratoire de commande
 *   Copyright (c) 1998-99
 *   Tous droits réservés
 *   version 2.0
 */

#define S_FUNCTION_NAME calcul_pp

#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 20); /* number of inputs */
    ssSetNumOutputs(       S, 9); /* number of outputs */

    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}
```

```
/*
 * md1Outputs - compute the outputs
 */

static void md1Outputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

y[0]=(1-(u[9]+u[13]+u[17])/u[19])*u[18]*u[9]-
((u[9]*u[0]+u[10]*u[1]+u[11]*u[2])*u[0]+(u[9]*u[3]+u[10]*u[4]+u[11]*u[5]
)*u[3]+(u[9]*u[6]+u[10]*u[7]+u[11]*u[8])*u[6])*u[9]-
((u[9]*u[0]+u[10]*u[1]+u[11]*u[2])*u[1]+(u[9]*u[3]+u[10]*u[4]+u[11]*u[5]
)*u[4]+(u[9]*u[6]+u[10]*u[7]+u[11]*u[8])*u[7])*u[12]-
((u[9]*u[0]+u[10]*u[1]+u[11]*u[2])*u[2]+(u[9]*u[3]+u[10]*u[4]+u[11]*u[5]
)*u[5]+(u[9]*u[6]+u[10]*u[7]+u[11]*u[8])*u[8])*u[15];
y[1]=(1-(u[9]+u[13]+u[17])/u[19])*u[18]*u[10]-
((u[9]*u[0]+u[10]*u[1]+u[11]*u[2])*u[0]+(u[9]*u[3]+u[10]*u[4]+u[11]*u[5]
)*u[3]+(u[9]*u[6]+u[10]*u[7]+u[11]*u[8])*u[6])*u[10]-
((u[9]*u[0]+u[10]*u[1]+u[11]*u[2])*u[1]+(u[9]*u[3]+u[10]*u[4]+u[11]*u[5]
)*u[4]+(u[9]*u[6]+u[10]*u[7]+u[11]*u[8])*u[7])*u[13]-
((u[9]*u[0]+u[10]*u[1]+u[11]*u[2])*u[2]+(u[9]*u[3]+u[10]*u[4]+u[11]*u[5]
)*u[5]+(u[9]*u[6]+u[10]*u[7]+u[11]*u[8])*u[8])*u[16];
y[2]=(1-(u[9]+u[13]+u[17])/u[19])*u[18]*u[11]-
((u[9]*u[0]+u[10]*u[1]+u[11]*u[2])*u[0]+(u[9]*u[3]+u[10]*u[4]+u[11]*u[5]
)*u[3]+(u[9]*u[6]+u[10]*u[7]+u[11]*u[8])*u[6])*u[11]-
((u[9]*u[0]+u[10]*u[1]+u[11]*u[2])*u[1]+(u[9]*u[3]+u[10]*u[4]+u[11]*u[5]
)*u[4]+(u[9]*u[6]+u[10]*u[7]+u[11]*u[8])*u[7])*u[14]-
((u[9]*u[0]+u[10]*u[1]+u[11]*u[2])*u[2]+(u[9]*u[3]+u[10]*u[4]+u[11]*u[5]
)*u[5]+(u[9]*u[6]+u[10]*u[7]+u[11]*u[8])*u[8])*u[17];
y[3]=(1-(u[9]+u[13]+u[17])/u[19])*u[18]*u[12]-
((u[12]*u[0]+u[13]*u[1]+u[14]*u[2])*u[0]+(u[12]*u[3]+u[13]*u[4]+u[14]*u[
5])*u[3]+(u[12]*u[6]+u[13]*u[7]+u[14]*u[8])*u[6])*u[9]-
((u[12]*u[0]+u[13]*u[1]+u[14]*u[2])*u[1]+(u[12]*u[3]+u[13]*u[4]+u[14]*u[
5])*u[4]+(u[12]*u[6]+u[13]*u[7]+u[14]*u[8])*u[7])*u[12]-
((u[12]*u[0]+u[13]*u[1]+u[14]*u[2])*u[2]+(u[12]*u[3]+u[13]*u[4]+u[14]*u[
5])*u[5]+(u[12]*u[6]+u[13]*u[7]+u[14]*u[8])*u[8])*u[15];
y[4]=(1-(u[9]+u[13]+u[17])/u[19])*u[18]*u[13]-
((u[12]*u[0]+u[13]*u[1]+u[14]*u[2])*u[0]+(u[12]*u[3]+u[13]*u[4]+u[14]*u[
5])*u[3]+(u[12]*u[6]+u[13]*u[7]+u[14]*u[8])*u[6])*u[10]-
((u[12]*u[0]+u[13]*u[1]+u[14]*u[2])*u[1]+(u[12]*u[3]+u[13]*u[4]+u[14]*u[
5])*u[4]+(u[12]*u[6]+u[13]*u[7]+u[14]*u[8])*u[7])*u[13]-
((u[12]*u[0]+u[13]*u[1]+u[14]*u[2])*u[2]+(u[12]*u[3]+u[13]*u[4]+u[14]*u[
5])*u[5]+(u[12]*u[6]+u[13]*u[7]+u[14]*u[8])*u[8])*u[16];
y[5]=(1-(u[9]+u[13]+u[17])/u[19])*u[18]*u[14]-
((u[12]*u[0]+u[13]*u[1]+u[14]*u[2])*u[0]+(u[12]*u[3]+u[13]*u[4]+u[14]*u[
5])*u[3]+(u[12]*u[6]+u[13]*u[7]+u[14]*u[8])*u[6])*u[11]-
((u[12]*u[0]+u[13]*u[1]+u[14]*u[2])*u[1]+(u[12]*u[3]+u[13]*u[4]+u[14]*u[
5])*u[4]+(u[12]*u[6]+u[13]*u[7]+u[14]*u[8])*u[7])*u[14]-
((u[12]*u[0]+u[13]*u[1]+u[14]*u[2])*u[2]+(u[12]*u[3]+u[13]*u[4]+u[14]*u[
5])*u[5]+(u[12]*u[6]+u[13]*u[7]+u[14]*u[8])*u[8])*u[17];
y[6]=(1-(u[9]+u[13]+u[17])/u[19])*u[18]*u[15]-
((u[15]*u[0]+u[16]*u[1]+u[17]*u[2])*u[0]+(u[15]*u[3]+u[16]*u[4]+u[17]*u[
5])*u[3]+(u[15]*u[6]+u[16]*u[7]+u[17]*u[8])*u[6])*u[9]-
((u[15]*u[0]+u[16]*u[1]+u[17]*u[2])*u[1]+(u[15]*u[3]+u[16]*u[4]+u[17]*u[
5])*u[4]+(u[15]*u[6]+u[16]*u[7]+u[17]*u[8])*u[7])*u[12]-
```

```
((u[15]*u[0]+u[16]*u[1]+u[17]*u[2])*u[2]+(u[15]*u[3]+u[16]*u[4]+u[17]*u[
5])*u[5]+(u[15]*u[6]+u[16]*u[7]+u[17]*u[8])*u[8])*u[15];
y[7]=(1-(u[9]+u[13]+u[17])/u[19])*u[18]*u[16]-
((u[15]*u[0]+u[16]*u[1]+u[17]*u[2])*u[0]+(u[15]*u[3]+u[16]*u[4]+u[17]*u[
5])*u[3]+(u[15]*u[6]+u[16]*u[7]+u[17]*u[8])*u[6])*u[10]-
((u[15]*u[0]+u[16]*u[1]+u[17]*u[2])*u[1]+(u[15]*u[3]+u[16]*u[4]+u[17]*u[
5])*u[4]+(u[15]*u[6]+u[16]*u[7]+u[17]*u[8])*u[7])*u[13]-
((u[15]*u[0]+u[16]*u[1]+u[17]*u[2])*u[2]+(u[15]*u[3]+u[16]*u[4]+u[17]*u[
5])*u[5]+(u[15]*u[6]+u[16]*u[7]+u[17]*u[8])*u[8])*u[16];
y[8]=(1-(u[9]+u[13]+u[17])/u[19])*u[18]*u[17]-
((u[15]*u[0]+u[16]*u[1]+u[17]*u[2])*u[0]+(u[15]*u[3]+u[16]*u[4]+u[17]*u[
5])*u[3]+(u[15]*u[6]+u[16]*u[7]+u[17]*u[8])*u[6])*u[11]-
((u[15]*u[0]+u[16]*u[1]+u[17]*u[2])*u[1]+(u[15]*u[3]+u[16]*u[4]+u[17]*u[
5])*u[4]+(u[15]*u[6]+u[16]*u[7]+u[17]*u[8])*u[7])*u[14]-
((u[15]*u[0]+u[16]*u[1]+u[17]*u[2])*u[2]+(u[15]*u[3]+u[16]*u[4]+u[17]*u[
5])*u[5]+(u[15]*u[6]+u[16]*u[7]+u[17]*u[8])*u[8])*u[17];


}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Calcul de la loi de commande de position (Loi_commande)

```
/*

 * LOI_COMMANDE : Calcul de la loi de commande (Tcom)

 *

 *   Entrées : a (paramètres estimés : masses)

 *             qrp (vitesse articulaire modifiée)

 *             qrpp-sd (accélération articulaire modifiée et comprenant
un contrôleur PD)

 *             qp (vitesse articulaire)

 *             vecteur de précalcul de sinus et de consinus de q
(position articulaire)

 *

 *   Sorties : 1-3 -> Couple de commande

 *

 *

 *   Par : Martin de Montigny

 *   UQTR : Laboratoire de commande

 *   Copyright (c) 1998-99

 *   Tous droits réservés

 *   version 2.0

 */


#define S_FUNCTION_NAME Loi_commande

#define l2L2 0.24
#define L2L2 0.64
#define l0l0 0.16
#define L1L2 0.64
#define l2L1 0.24
#define l1l1 0.16
#define l2l2 0.09
#define L1L1 0.64
#define l1L1 0.32
#define l1 0.4
#define L2 0.8
#define l0 0.4
```

```
#define l2 0.3
#define L1 0.8
#define L3 0.6
#define L4 0.1




#include "simstruc.h"

#include "math.h"


/*

 * mdlInitializeSizes - initialize the sizes array

 */

static void mdlInitializeSizes(SimStruct *S)

{

    ssSetNumContStates(    S, 0);   /* number of continuous states
*/

    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/

    ssSetNumInputs(S,21);
    ssSetNumOutputs(S,3);

    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/

    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/

    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/

    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */

    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/

    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/

}
```

```
/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}



/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}


/*
 * mdlOutputs - compute the outputs
 */


static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{


y[0]=(-1.*u[5]*u[9]*l2L1-1.*u[4]*u[9]*l2L1-1.*u[3]*u[11]*l2L1-
1.*u[5]*u[10]*l2L1-1.*u[4]*u[10]*12L1-1.*u[3]*u[10]*l2L1-
1.*u[5]*u[11]*l2L1-
1.*u[4]*u[11]*l2L1)*u[2]*u[14]+(l0l0*u[6]+9.81000000000*u[16]*l0)*u[0]+(
u[6]*l1l1+u[6]*L1L1+2.*u[6]*u[19]*l1L1+u[7]*u[19]*l1L1+l1l1*u[7]-
1.*u[10]*u[18]*l1L1*u[3]-1.*u[4]*u[9]*u[18]*l1L1-
```

```
1.*u[4]*u[10]*u[18]*l1L1+9.81000000000*u[13]*l1+9.81000000000*u[16]*L1)*
u[1]+(-1.*u[4]*u[9]*u[18]*L1L2-1.*u[3]*u[17]*u[11]*l2L2-
1.*u[3]*u[10]*u[18]*L1L2+u[7]*u[15]*l2L1+u[7]*u[19]*L1L2+2.*u[7]*u[20]*l
2L2+2.*u[6]*u[15]*l2L1+2.*u[6]*u[19]*L1L2+2.*u[6]*u[20]*l2L2+u[6]*L1L1-
1.*u[4]*u[10]*u[18]*L1L2-
1.*u[5]*u[17]*u[9]*l2L2+u[8]*u[15]*l2L1+u[8]*u[20]*l2L2-
1.*u[5]*u[17]*u[11]*l2L2-
1.*u[5]*u[17]*u[10]*l2L2+9.81000000000*u[12]*l2+9.81000000000*u[13]*L2+u
[6]*L2L2+u[7]*l2l2+u[7]*L2L2+u[6]*l2l2+9.81000000000*u[16]*L1+l2l2*u[8]-
1.*u[17]*u[11]*l2L2*u[4])*u[2];
y[1]=u[2]*u[3]*u[9]*u[14]*l2L1+(u[6]*u[19]*l1L1+u[6]*l1l1+l1l1*u[7]+u[9]
*u[18]*l1L1*u[3]+9.81000000000*u[13]*l1)*u[1]+(2.*u[6]*u[20]*l2L2+u[6]*L
2L2+u[6]*u[19]*L1L2+u[6]*l2l2+u[6]*u[15]*l2L1+2.*u[7]*u[20]*l2L2+u[7]*L2
L2+u[7]*l2l2+u[8]*u[20]*l2L2+l2l2*u[8]+u[3]*u[9]*u[18]*L1L2-
1.*u[5]*u[17]*u[10]*l2L2-1.*u[3]*u[17]*u[11]*l2L2-
1.*u[17]*u[11]*l2L2*u[4]-1.*u[5]*u[17]*u[9]*l2L2-
1.*u[5]*u[17]*u[11]*l2L2+9.81000000000*u[13]*L2+9.81000000000*u[12]*l2)*
u[2];
y[2]=u[2]*u[3]*u[9]*u[14]*l2L1+(u[6]*u[20]*l2L2+u[6]*l2l2+u[6]*u[15]*l2L
1+u[7]*u[20]*l2L2+u[7]*l2l2+l2l2*u[8]+u[3]*u[17]*u[9]*l2L2+9.810000000000
*u[12]*l2+u[3]*u[17]*u[10]*l2L2+u[4]*u[17]*u[9]*l2L2+u[4]*u[17]*u[10]*l2
L2)*u[2];




}



/*

 * mdlUpdate - perform action at major integration time step

 */



static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)

{

}



/*

 * mdlDerivatives - compute the derivatives

 */

static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)

{
```

```
}



/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}


#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Calcul de la matrice des signaux W (Calcul_w)

```
/*
 * CALCUL_W : Calcul de w (pour calculer T filtré)
 *
 *   Entrées : qp (vitesse articulaire)
 *             g (constante gravitationnelle)
```

```
 *             vecteur de précalcul de sinus et de cosinus de q
(position articulaire)

 *             lambda (fréquence du filtre)

 *

 *   Sorties : 1-9 -> Partie 1 de la matrice W (à filtrer)

 *             10-18 -> Partie 2 de la matrice W

 *

 *

 *   Par : Martin de Montigny

 *   UQTR : Laboratoire de commande

 *   Opal-rt

 *   Copyright (c) 1998-99

 *   Tous droits réservés

 *   version 2.0

 */


#define S_FUNCTION_NAME Calcul_w

#define l2L2 0.24
#define L2L2 0.64
#define l0l0 0.16
#define L1L2 0.64
#define l2L1 0.24
#define l1l1 0.16
#define l2l2 0.09
#define L1L1 0.64
#define l1L1 0.32
#define l1 0.4
#define L2 0.8
#define l0 0.4
#define l2 0.3
#define L1 0.8
#define L3 0.6
#define L4 0.1
```

```
#include "simstruc.h"

#include "math.h"


/*

 * mdlInitializeSizes - initialize the sizes array

 */

static void mdlInitializeSizes(SimStruct *S)

{

    ssSetNumContStates(    S, 0);   /* number of continuous states
*/

    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/

    ssSetNumInputs(S,13);
    ssSetNumOutputs(S,27);

    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/

    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/

    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/

    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */

    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/

    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/

}


/*

 * mdlInitializeSampleTimes - initialize the sample times array

 */

static void mdlInitializeSampleTimes(SimStruct *S)

{
```

```
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);

    ssSetOffsetTime(S, 0, 0.0);

}



/*

 * mdlInitializeConditions - initialize the states

 */

static void mdlInitializeConditions(double *x0, SimStruct *S)

{

}



/*

 * mdlOutputs - compute the outputs

 */



static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)

{



y[0]=0.981E1*u[7]*l0;
y[1]=0.981E1*u[4]*l1+0.981E1*u[7]*L1;
y[2]=0.981E1*u[4]*L2+0.981E1*u[3]*l2+0.981E1*u[7]*L1;
y[3]=0.0;
y[4]=u[0]*u[0]*u[9]*l1L1+u[0]*u[1]*u[9]*l1L1+0.981E1*u[4]*l1;
y[5]=(u[0]*u[1]*l2L1+u[0]*u[2]*l2L1+u[0]*u[0]*l2L1)*u[5]+u[0]*u[0]*u[9]*
L1L2+0.981E1*u[4]*L2+u[0]*u[1]*u[9]*L1L2+0.981E1*u[3]*l2;
y[6]=0.0;
y[7]=0.0;
y[8]=(u[0]*u[1]*l2L1+u[0]*u[2]*l2L1+u[0]*u[0]*l2L1)*u[5]+u[8]*u[0]*u[0]*
l2L2+2.0*u[0]*u[8]*u[1]*l2L2+u[8]*u[2]*l2L2*u[1]+u[0]*u[8]*u[2]*l2L2+u[8
]*u[1]*u[1]*l2L2+0.981E1*u[3]*l2;
y[9]=-l0l0*u[0];
y[10]=(-l1l1-L1L1-2.0*u[10]*l1L1)*u[0]+(-u[10]*l1L1-l1l1)*u[1];
y[11]=(-2.0*u[11]*l2L2-L2L2-2.0*u[10]*L1L2-2.0*u[6]*l2L1-l2l2-
L1L1)*u[0]+(-2.0*u[11]*l2L2-L2L2-u[10]*L1L2-l2l2-u[6]*l2L1)*u[1]+(-
u[11]*l2L2-l2l2-u[6]*l2L1)*u[2];
y[12]=0.0;
```

```
y[13]=(-u[10]*l1L1-l1l1)*u[0]-u[1]*l1l1;
y[14]=(-2.0*u[11]*l2L2-L2L2-u[10]*L1L2-l2l2-u[6]*l2L1)*u[0]+(-
2.0*u[11]*l2L2-L2L2-l2l2)*u[1]+(-u[11]*l2L2-l2l2)*u[2];
y[15]=0.0;
y[16]=0.0;
y[17]=(-u[11]*l2L2-l2l2-u[6]*l2L1)*u[0]+(-u[11]*l2L2-l2l2)*u[1]-
l2l2*u[2];
y[18]=u[12]*l0l0*u[0];
y[19]=u[12]*((l1l1+L1L1+2.0*u[10]*l1L1)*u[0]+(u[10]*l1L1+l1l1)*u[1]);
y[20]=u[12]*((2.0*u[11]*l2L2+L2L2+2.0*u[10]*L1L2+2.0*u[6]*l2L1+l2l2+L1L1
)*u[0]+(2.0*u[11]*l2L2+L2L2+u[10]*L1L2+l2l2+u[6]*l2L1)*u[1]+(u[11]*l2L2+
l2l2+u[6]*l2L1)*u[2]);
y[21]=0.0;
y[22]=u[12]*((u[10]*l1L1+l1l1)*u[0]+u[1]*l1l1);
y[23]=u[12]*((2.0*u[11]*l2L2+L2L2+u[10]*L1L2+l2l2+u[6]*l2L1)*u[0]+(2.0*u
[11]*l2L2+L2L2+l2l2)*u[1]+(u[11]*l2L2+l2l2)*u[2]);
y[24]=0.0;
y[25]=0.0;
y[26]=u[12]*((u[11]*l2L2+l2l2+u[6]*l2L1)*u[0]+(u[11]*l2L2+l2l2)*u[1]+l2l
2*u[2]);



}



/*

 * mdlUpdate - perform action at major integration time step

 */



static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)

{

}



/*

 * mdlDerivatives - compute the derivatives

 */

static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)

{

}
```

```
/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}


#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Bloc d'appel des deux routines de calcul de la dynamique inverse selon la formulation de Lagrange-Euler

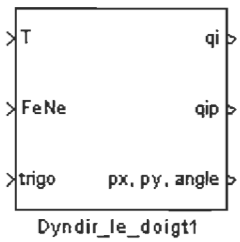

Figure C. 15 : Calcul de la dynamique inverse selon la formulation de Lagrange-Euler

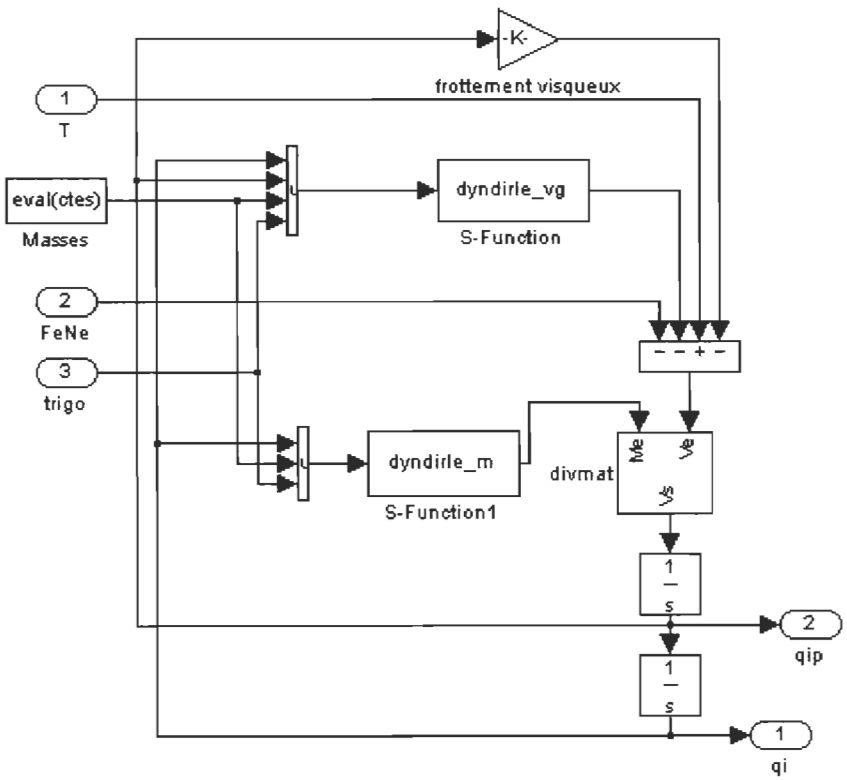

Figure C. 16 : Intérieur du bloc de calcul de la dynamique inverse selon la formulation de Lagrange-Euler

Routine de calcul de la dynamique inverse selon la formulation de Lagrange-Euler pour le calcul des termes de gravité, forces centrifuges et de Coriolis (dyndirle_vg)

```
/*
 * DYNDIRLE_VG : Dynamique directe du doigt selon la méthode de
Lagrange-Euler :
 *               Sous-fonction de calcul du vecteur de termes de vitesse
et de
 *               coriolis et de gravité.
 *
 *   Entrées : q (position articulaire)
 *             qp (vitesse articulaire)
 *             masses des membres
 *             vecteur des termes trigonométriques
 *
 *   Sorties : l-M -> Couple représentant les termes de vitesse/coriolis
et de gravité
 *
 *   Par : Martin de Montigny
 *   UQTR : Laboratoire de commande
 *   Copyright (c) février 1999
 *   Tous droits réservés
 *   version 1.1
 */

#define S_FUNCTION_NAME dyndirle_vg

#include "simstruc.h"
#include "math.h"
/* Définition des constantes */
#define l2L2 0.24
#define L2L2 0.64
#define l0l0 0.16
#define L1L2 0.64
#define l2L1 0.24
#define l1l1 0.16
#define l2l2 0.09
#define L1L1 0.64
#define l1L1 0.32
#define l1 0.4
#define L2 0.8
#define l0 0.4
#define l2 0.3
#define L1 0.8
#define L3 0.6
#define L4 0.1



/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
```

```
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
/* nombre d'entrées et de sorties */
    ssSetNumInputs(S,18);
    ssSetNumOutputs(S,3);
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
/* Algorithme */
y[0]=-2.0*u[15]*u[7]*u[3]*u[4]*l1L1-u[15]*u[7]*u[4]*u[4]*l1L1-
2.0*u[14]*u[8]*u[4]*u[5]*l2L2-2.0*u[15]*u[8]*u[3]*u[4]*L1L2-
u[15]*u[8]*u[4]*u[4]*L1L2-u[8]*u[4]*u[4]*u[11]*l2L1-
u[8]*u[5]*u[5]*u[11]*l2L1-u[14]*u[8]*u[5]*u[5]*l2L2-
2.0*u[14]*u[8]*u[3]*u[5]*l2L2-2.0*u[8]*u[3]*u[4]*u[11]*l2L1-
2.0*u[8]*u[4]*u[5]*u[11]*l2L1-
2.0*u[8]*u[3]*u[5]*u[11]*l2L1+(0.981E1*u[7]*u[10]*l1+0.981E1*u[8]*u[10]*
L2+0.981E1*u[6]*u[13]*l0+0.981E1*u[8]*u[9]*l2+0.981E1*u[8]*u[13]*L1+0.98
1E1*u[7]*u[13]*L1);
y[1]=-2.0*u[14]*u[8]*u[4]*u[5]*l2L2-u[14]*u[8]*u[5]*u[5]*l2L2-
2.0*u[14]*u[8]*u[3]*u[5]*l2L2+u[8]*u[3]*u[3]*u[11]*l2L1+u[8]*u[15]*u[3]*
```

```
u[3]*L1L2+u[7]*u[15]*u[3]*u[3]*l1L1+(0.981E1*u[7]*u[10]*l1+0.981E1*u[8]*
u[10]*L2+0.981E1*u[8]*u[9]*l2);
y[2]=u[8]*u[14]*u[3]*u[3]*l2L2+2.0*u[8]*u[14]*u[3]*u[4]*l2L2+u[8]*u[14]*
u[4]*u[4]*l2L2+u[8]*u[3]*u[3]*u[11]*l2L1+(0.981E1*u[8]*u[9]*l2);


}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Routine de calcul de la dynamique inverse selon la formulation de Lagrange-Euler pour le calcul des colonnes de la matrice des masses (dyndirle_m)

```
/*

 * DYNDIRLE2_M : Dynamique directe du doigt selon la méthode de
Lagrange-Euler :

 *               Sous-fonction de calcul de la matrice des masses
```

```
 *
 *   Entrées : position articulaire
 *             masses de l'articulation
 *             vecteur de calcul des sinus et cosinus
 *
 *
 *   Sortie : 1-M^2 -> matrice des masses/inerties
 *
 *   Par : Martin de Montigny
 *   UQTR : Laboratoire de commande
 *   Copyright (c) février 1999
 *   Tous droits réservés
 *   version 1.0
 */


#define S_FUNCTION_NAME dyndirle_m


#include "simstruc.h"

#include "math.h"

/*définition des constantes*/

#define l2L2 0.24
#define L2L2 0.64
#define l0l0 0.16
#define L1L2 0.64
#define l2L1 0.24
#define l1l1 0.16
#define l2l2 0.09
#define L1L1 0.64
#define l1L1 0.32
#define l1 0.4
#define L2 0.8
#define l0 0.4
#define l2 0.3
#define L1 0.8
#define L3 0.6
#define L4 0.1
```

```
/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
/*nombre d'entrées et de sorties*/
    ssSetNumInputs(S,15);
    ssSetNumOutputs(S,9);
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}
/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
```

```
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);

    ssSetOffsetTime(S, 0, 0.0);

}



/*

 * mdlInitializeConditions - initialize the states

 */

static void mdlInitializeConditions(double *x0, SimStruct *S)

{

}



/*

 * mdlOutputs - compute the outputs

 */



static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)

{

/*équations*/

y[0]=2.0*u[5]*u[14]*l2L2+u[5]*L2L2+u[3]*l0l0+2.0*u[5]*u[13]*L1L2+2.0*u[5
]*u[9]*l2L1+u[4]*l1l1+u[5]*l2l2+u[4]*L1L1+u[5]*L1L1+2.0*u[4]*u[13]*l1L1;
y[3]=2.0*u[5]*u[14]*l2L2+u[5]*L2L2+u[4]*u[13]*l1L1+u[5]*u[13]*L1L2+u[4]*
l1l1+u[5]*l2l2+u[5]*u[9]*l2L1;
y[4]=2.0*u[5]*u[14]*l2L2+u[5]*L2L2+u[4]*l1l1+u[5]*l2l2;
y[6]=u[5]*u[14]*l2L2+u[5]*l2l2+u[5]*u[9]*l2L1;
y[7]=u[5]*u[14]*l2L2+u[5]*l2l2;
y[8]=u[5]*l2l2;
y[1]=y[3];
y[2]=y[6];
y[5]=y[7];
```

```
}


/*
 * mdlUpdate - perform action at major integration time step
 */


static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}


/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}




/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}


#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
```

```
#else

#include "cg_sfun.h"       /* Code generation registration function */

#endif
```

Librairie des blocs non classés (Divers)

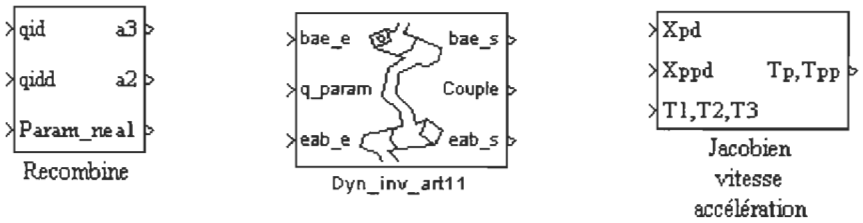

Figure C. 17 : Librairie des blocs non classés

Fonction de recombinaison (Recombine)

```
/*
 * RECOMBINE Fonction de recombinaison des données de la dynamique
inverse
 *
 *
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   UQTR
 *   Copyright (c) aout 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME recombine

#include "simstruc.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
```

```
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 133);/* number of inputs         */
    ssSetNumOutputs(       S, 156);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
   /* Recombinaison_des_entrées */
y[0]=u[0];
y[1]=u[3];
y[2]=u[6];
y[3]=u[11];
y[4]=u[12];
y[5]=u[13];
y[6]=u[26];
y[7]=u[27];
y[8]=u[28];
y[9]=u[29];
y[10]=u[30];
y[11]=u[31];
y[12]=u[32];
y[13]=u[33];
```

```
y[14]=u[34];
y[15]=u[71];
y[16]=u[72];
y[17]=u[73];
y[18]=u[74];
y[19]=u[75];
y[20]=u[76];
y[21]=u[77];
y[22]=u[78];
y[23]=u[79];
y[24]=u[80];
y[25]=u[81];
y[26]=u[82];
y[27]=u[83];
y[28]=u[84];
y[29]=u[85];
y[30]=u[86];
y[31]=u[87];
y[32]=u[88];
y[33]=u[116];
y[34]=u[117];
y[35]=u[118];
y[36]=u[119];
y[37]=u[120];
y[38]=u[121];
y[39]=u[1];
y[40]=u[4];
y[41]=u[7];
y[42]=u[14];
y[43]=u[15];
y[44]=u[16];
y[45]=u[35];
y[46]=u[36];
y[47]=u[37];
y[48]=u[38];
y[49]=u[39];
y[50]=u[40];
y[51]=u[41];
y[52]=u[42];
y[53]=u[43];
y[54]=u[80];
y[55]=u[81];
y[56]=u[82];
y[57]=u[83];
y[58]=u[84];
y[59]=u[85];
y[60]=u[86];
y[61]=u[87];
y[62]=u[88];
y[63]=u[89];
y[64]=u[90];
y[65]=u[91];
y[66]=u[92];
y[67]=u[93];
y[68]=u[94];
y[69]=u[95];
y[70]=u[96];
```

```
y[71]=u[97];
y[72]=u[119];
y[73]=u[120];
y[74]=u[121];
y[75]=u[122];
y[76]=u[123];
y[77]=u[124];
y[78]=u[2];
y[79]=u[5];
y[80]=u[8];
y[81]=u[17];
y[82]=u[18];
y[83]=u[19];
y[84]=u[44];
y[85]=u[45];
y[86]=u[46];
y[87]=u[47];
y[88]=u[48];
y[89]=u[49];
y[90]=u[50];
y[91]=u[51];
y[92]=u[52];
y[93]=u[89];
y[94]=u[90];
y[95]=u[91];
y[96]=u[92];
y[97]=u[93];
y[98]=u[94];
y[99]=u[95];
y[100]=u[96];
y[101]=u[97];
y[102]=u[98];
y[103]=u[99];
y[104]=u[100];
y[105]=u[101];
y[106]=u[102];
y[107]=u[103];
y[108]=u[104];
y[109]=u[105];
y[110]=u[106];
y[111]=u[122];
y[112]=u[123];
y[113]=u[124];
y[114]=u[125];
y[115]=u[126];
y[116]=u[127];


}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}
```

```
/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Fonction de calcul de la dynamique inverse articulaire compilée en deux parties (Dyn_inv_art)

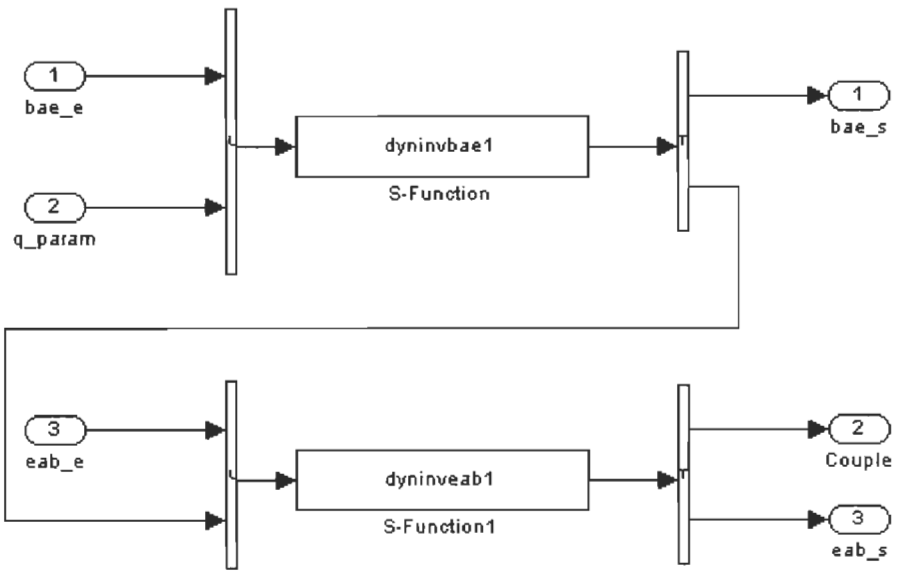

Figure C. 18 : Intérieur du bloc de calcul de la dynamique inverse articulaire compilé en deux parties

Nous présentons ci-dessous le résultat de la génération de code d'un manipulateur à trois membres (le doigt 1 présenté dans l'étude de cas) sous la formulation de Newton-Euler. Le premier code effectue l'itération de la base à l'effecteur tandis que le deuxième code calcule l'itération de l'effecteur à la base.

(dyninvbae1)

```
/*
 * DYNINV_BAE Calcul de la dynamique inverse articulaire; itération
 *                    base à effecteur.
 *
 *
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   Copyright (c) juillet 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME dyninv_bae

#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 48);/* number of inputs         */
    ssSetNumOutputs(       S, 30);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
```

```
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
   int i;

   /*Propagation de la dynamique (base à effecteur)*/
   /*wim1=u[0]->u[2]*/
   /*wpim1=u[3]->u[5]*/
   /*vpim1=u[6]->u[8]*/

   /*Variables articulaires, constantes et cinématique directe*/
   /*qid=u[9];*/
   /*qidd=u[10];*/
   /*masse=u[11];*/
   /*Pcidi=u[12]->u[14]*/
   /*tenseur=u[15]->u[23]*/
   /*Rivim1=u[24]->u[32]*/
   /*Rpi1vi=u[33]->u[41]*/
   /*Pidim1=u[42]->u[44]*/
   /*Pip1di=u[45]->u[47]*/

   /* Calcul de vitesse de rotation du referentiel de l'articulation */
   y[0]=u[24]*u[0]+u[27]*u[1]+u[30]*u[2];
   y[1]=u[25]*u[0]+u[28]*u[1]+u[31]*u[2];
   y[2]=u[29]*u[1]+u[32]*u[2]+u[9];


   /* Calcul d'acceleration rotative du referentiel de l'articulation */

y[3]=u[24]*u[3]+u[27]*u[4]+u[30]*u[5]+(u[25]*u[0]+u[28]*u[1]+u[31]*u[2])
*u[9];
   y[4]=u[25]*u[3]+u[28]*u[4]+u[31]*u[5]-
(u[24]*u[0]+u[27]*u[1]+u[30]*u[2])*u[9];
   y[5]=u[29]*u[4]+u[32]*u[5]+u[10];


   /* Calcul d'acceleration lineaire du referentiel de l'articulation */
```

```
   y[6]=u[24]*(-u[1]*u[1]*u[42]-
u[2]*u[2]*u[42]+u[6])+u[27]*(u[5]*u[42]+u[0]*u[1]*u[42]+u[7])+u[30]*(-
u[4]*u[42]+u[0]*u[2]*u[42]+u[8]);
   y[7]=u[25]*(-u[1]*u[1]*u[42]-
u[2]*u[2]*u[42]+u[6])+u[28]*(u[5]*u[42]+u[0]*u[1]*u[42]+u[7])+u[31]*(-
u[4]*u[42]+u[0]*u[2]*u[42]+u[8]);
   y[8]=u[29]*(u[5]*u[42]+u[0]*u[1]*u[42]+u[7])+u[32]*(-
u[4]*u[42]+u[0]*u[2]*u[42]+u[8]);


   /* Calcul des forces et couples inertiels */
   y[9]=u[11]*((u[13]*y[1]+u[14]*y[2])*y[0]+y[4]*u[14]-y[5]*u[13]-
y[1]*y[1]*u[12]-y[2]*y[2]*u[12]+y[6]);
   y[10]=u[11]*(-y[0]*y[0]*u[13]+y[0]*y[1]*u[12]+y[5]*u[12]-
y[3]*u[14]+y[2]*(y[1]*u[14]-y[2]*u[13])+y[7]);
   y[11]=u[11]*(-y[0]*y[0]*u[14]+y[0]*y[2]*u[12]+y[3]*u[13]-y[4]*u[12]-
y[1]*(y[1]*u[14]-y[2]*u[13])+y[8]);
   y[12]=0;
   y[13]=0;
   y[14]=0;



   /* Sorties */

   /* Passage de la position de la masse à l'itération effecteur à base
(pcidi)*/
      y[15]=u[12];
      y[16]=u[13];
      y[17]=u[14];

      /* Passage de Rivim1 et Pidim1 pour itération de effecteur à base
*/

      /*Rip1vi*/
      y[18]=u[33];
      y[19]=u[34];
      y[20]=u[35];
      y[21]=u[36];
      y[22]=u[37];
      y[23]=u[38];
      y[24]=u[39];
      y[25]=u[40];
      y[26]=u[41];
      /*Pip1di*/
      y[27]=u[45];
      y[28]=u[46];
      y[29]=u[47];

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
```

```
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

(dyninveab1)

```
/*
 * DYNINV_EAB Calcul de la dynamique inverse du manipulateur; itération
de l'effecteur
 *            à la base.
 *
 *
 *    entrées : fip1, nip1, Fi, Ni, pcidi, Rip1vi, Pip1di.
 *
 *    sorties : Ti, fi, ni.
 *
 *    Signification des variables :
 *
 *      fip1 : force externe exercée sur le membre suivant par le membre
actuel
 *      nip1 : couple externe exercé sur le membre suivant par le membre
actuel
 *      Fi : force dù à l'inertie
 *      Ni : couple dù à l'inertie
 *      pcidi : position de la masse par rapport au référentiel du
membre
 *      Rip1vi : matrice de rotation du référentiel du membre suivant
 *               par rapport au référentiel du membre actuel
 *      Pip1di : vecteur de position du référentiel du membre suivant
 *               par rapport au référentiel du membre actuel
 *
 *      Ti : couple articulaire
 *      fi : force externe exercée sur le membre actuel par le membre
précédent
```

```
 *      ni : couple externe exercé sur le membre actuel par le membre
précédent
 *
 *   Par : Martin de Montigny
 *   Opal-rt
 *   Copyright (c) juillet 1998
 *   Tous droits réservés
 *   version 1.0
 */

#define S_FUNCTION_NAME dyninv_eab

#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states
*/
    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/
    ssSetNumInputs(        S, 27);/* number of inputs         */
    ssSetNumOutputs(       S, 7);   /* number of outputs     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/
    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
    ssSetNumRWork(         S, 0);   /* number of real work vector
elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector
elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector
elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}
```

```
/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

   /* Entrées
   fip1 = u[0]->u[2]
   nip1 = u[3]->u[5]
   Fi = u[6]->u[8]
   Ni = u[9]->u[11]
   pcidi = u[12]->u[14]
   Rip1vi = u[15]->u[23]
   Pip1di = u[24]->u[26]
   */

   /* Sorties
   T = y[0]
   fi = y[1]->y[3]
   ni = y[4]->y[6]
   */

   /* Calcul des forces et couples externes* */
   y[1]=u[15]*u[0]+u[16]*u[1]+u[6];
   y[2]=u[18]*u[0]+u[19]*u[1]+u[7];
   y[3]=u[23]*u[2]+u[8];
   y[4]=u[15]*u[3]+u[16]*u[4]+u[13]*u[8]-u[14]*u[7];
   y[5]=u[18]*u[3]+u[19]*u[4]+u[14]*u[6]-u[12]*u[8]-u[24]*u[23]*u[2];
   y[6]=u[23]*u[5]+u[12]*u[7]-u[13]*u[6]+u[24]*(u[18]*u[0]+u[19]*u[1]);


   /* Sorties */
   /* Calcul du couple articulaire* */
   y[0]=y[6];
}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
```

```
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Routine de conversion de vitesse/accélération galliléennes en vitesse/accélération articulaires (Jacobien vitesse accélération)

```
/*

 * Jacobien_vitesse_acceleration : Calcul de la vitesse et de
l'accélération articulaire en fonction

 *                                 de la vitesse et de l'accélération
cartésienne

 *

 *   Entrées : Vitesse cartésienne

 *             Accélération cartésienne

 *             Position articulaire

 *

 *   Sortie  : 1 à 2M    -> Vitesses et accélérations articulaires

 *

 *

 *   Par : Martin de Montigny

 *   UQTR : Laboratoire de commande

 *   Copyright (c) 1998-99

 *   Tous droits réservés

 *   version 2.0

 */
```

```
#define S_FUNCTION_NAME jacobien_vitacc

#define l2L2 0.24
#define L2L2 0.64
#define l0l0 0.16
#define L1L2 0.64
#define l2L1 0.24
#define l1l1 0.16
#define l2l2 0.09
#define L1L1 0.64
#define l1L1 0.32
#define l1 0.4
#define L2 0.8
#define l0 0.4
#define l2 0.3
#define L1 0.8
#define L3 0.6
#define L4 0.1




#include "simstruc.h"

#include "math.h"



/*

 * mdlInitializeSizes - initialize the sizes array

 */

static void mdlInitializeSizes(SimStruct *S)

{

    ssSetNumContStates(    S, 0);   /* number of continuous states
*/

    ssSetNumDiscStates(    S, 0);   /* number of discrete states
*/

    ssSetNumInputs(S,9);
    ssSetNumOutputs(S,6);

    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag
*/

    ssSetNumSampleTimes(   S, 1);   /* number of sample times
*/

    ssSetNumSFcnParams(    S, 0);   /* number of input arguments
*/
```

```
    ssSetNumRWork(          S, 0);   /* number of real work vector
elements   */

    ssSetNumIWork(          S, 0);   /* number of integer work vector
elements*/

    ssSetNumPWork(          S, 0);   /* number of pointer work vector
elements*/

}


/*

 * mdlInitializeSampleTimes - initialize the sample times array

 */

static void mdlInitializeSampleTimes(SimStruct *S)

{

    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);

    ssSetOffsetTime(S, 0, 0.0);

}




/*

 * mdlInitializeConditions - initialize the states

 */

static void mdlInitializeConditions(double *x0, SimStruct *S)

{

}


/*

 * mdlOutputs - compute the outputs

 */
```

```
static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)

{



double temp1;

temp1=sin(u[7]);
temp1=temp1*temp1;

y[0]=1.000*cos(u[6]+u[7])/L1/sin(u[7])*u[0]+1.000*sin(u[6]+u[7])/L1/sin(
u[7])*u[1]+(-1.000*L4*cos(u[8])-1.000*L3*sin(u[8]))/L1/sin(u[7])*u[2];
y[1]=(-1.000*cos(u[6]+u[7])*L2-
1.000*cos(u[6])*L1)/sin(u[7])/L2/L1*u[0]+(-1.000*sin(u[6]+u[7])*L2-
1.000*sin(u[6])*L1)/sin(u[7])/L2/L1*u[1]+(1.000*L2*L4*cos(u[8])+1.000*L2
*L3*sin(u[8])+1.000*L1*L4*cos(u[7]+u[8])+1.000*L1*L3*sin(u[7]+u[8]))/sin
(u[7])/L2/L1*u[2];
y[2]=1.000*cos(u[6])/L2/sin(u[7])*u[0]+1.000*sin(u[6])/L2/sin(u[7])*u[1]
+(-1.000*L4*cos(u[7]+u[8])-1.000*L3*sin(u[7]+u[8])-
1.000*L2*sin(u[7]))/L2/sin(u[7])*u[2];
y[3]=(-1.000*sin(u[6]+u[7])/L1/sin(u[7])*y[0]+(-
1.000*sin(u[6]+u[7])/L1/sin(u[7])-
1.000*cos(u[6]+u[7])/L1/temp1*cos(u[7]))*y[1])*u[0]+(1.000*cos(u[6]+u[7]
)/L1/sin(u[7])*y[0]+(1.000*cos(u[6]+u[7])/L1/sin(u[7])-
1.000*sin(u[6]+u[7])/L1/temp1*cos(u[7]))*y[1])*u[1]+(-1.*(-
1.000*L4*cos(u[8])-
1.000*L3*sin(u[8]))/L1/temp1*cos(u[7])*y[1]+(1.000*L4*sin(u[8])-
1.000*L3*cos(u[8]))/L1/sin(u[7])*y[2])*u[2]+1.000*cos(u[6]+u[7])/L1/sin(
u[7])*u[3]+1.000*sin(u[6]+u[7])/L1/sin(u[7])*u[4]+(-1.000*L4*cos(u[8])-
1.000*L3*sin(u[8]))/L1/sin(u[7])*u[5];
y[4]=((1.000*sin(u[6]+u[7])*L2+1.000*sin(u[6])*L1)/sin(u[7])/L2/L1*y[0]+
(1.000*sin(u[6]+u[7])/L1/sin(u[7])-1.*(-1.000*cos(u[6]+u[7])*L2-
1.000*cos(u[6])*L1)/temp1/L2/L1*cos(u[7]))*y[1])*u[0]+((-
1.000*cos(u[6]+u[7])*L2-1.000*cos(u[6])*L1)/sin(u[7])/L2/L1*y[0]+(-
1.000*cos(u[6]+u[7])/L1/sin(u[7])-1.*(-1.000*sin(u[6]+u[7])*L2-
1.000*sin(u[6])*L1)/temp1/L2/L1*cos(u[7]))*y[1])*u[1]+(((-
1.000*L1*L4*sin(u[7]+u[8])+1.000*L1*L3*cos(u[7]+u[8]))/sin(u[7])/L2/L1-
1.*(1.000*L2*L4*cos(u[8])+1.000*L2*L3*sin(u[8])+1.000*L1*L4*cos(u[7]+u[8
])+1.000*L1*L3*sin(u[7]+u[8]))/temp1/L2/L1*cos(u[7]))*y[1]+(-
1.000*L2*L4*sin(u[8])+1.000*L2*L3*cos(u[8])-
1.000*L1*L4*sin(u[7]+u[8])+1.000*L1*L3*cos(u[7]+u[8]))/sin(u[7])/L2/L1*y
[2])*u[2]+(-1.000*cos(u[6]+u[7])*L2-
1.000*cos(u[6])*L1)/sin(u[7])/L2/L1*u[3]+(-1.000*sin(u[6]+u[7])*L2-
1.000*sin(u[6])*L1)/sin(u[7])/L2/L1*u[4]+(1.000*L2*L4*cos(u[8])+1.000*L2
*L3*sin(u[8])+1.000*L1*L4*cos(u[7]+u[8])+1.000*L1*L3*sin(u[7]+u[8]))/sin
(u[7])/L2/L1*u[5];
y[5]=(-1.000*sin(u[6])/L2/sin(u[7])*y[0]-
1.000*cos(u[6])/L2/temp1*cos(u[7])*y[1])*u[0]+(1.000*cos(u[6])/L2/sin(u[
7])*y[0]-
1.000*sin(u[6])/L2/temp1*cos(u[7])*y[1])*u[1]+(((1.000*L4*sin(u[7]+u[8])
-1.000*L3*cos(u[7]+u[8])-1.000*L2*cos(u[7]))/L2/sin(u[7])-1.*(-
1.000*L4*cos(u[7]+u[8])-1.000*L3*sin(u[7]+u[8])-
1.000*L2*sin(u[7]))/L2/temp1*cos(u[7]))*y[1]+(1.000*L4*sin(u[7]+u[8])-
1.000*L3*cos(u[7]+u[8]))/L2/sin(u[7])*y[2])*u[2]+1.000*cos(u[6])/L2/sin(
```

```
u[7])*u[3]+1.000*sin(u[6])/L2/sin(u[7])*u[4]+(-1.000*L4*cos(u[7]+u[8])-
1.000*L3*sin(u[7]+u[8])-1.000*L2*sin(u[7]))/L2/sin(u[7])*u[5];




}


/*
 * mdlUpdate - perform action at major integration time step
 */


static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}


/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}




/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}
```

```
#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */

#include "simulink.c"      /* MEX-file interface mechanism */

#else

#include "cg_sfun.h"       /* Code generation registration function */

#endif
```

Annexe D

Programmes CINDIR et DYNAM

Programme CINDIR de calcul de la cinématique directe d'un manipulateur rigide série à N joints (version adaptée au générateur de code sous la forme de Lagrange-Euler

```
function
[Tivim1,Riv0,Pid0,JVid0]=cindir(M,N,Alphaim1,Aim1,Di,Thetai,Ki);
% CINDIR Détermine les matrices de transformation entre les référentiels
%        adjacents, les matrices de rotation et les vecteurs de position
%        des référentiels par rapport à la base, et forme les Jacobiens
%        de vitesse de chaque référentiel par rapport au référentiel de
%        base (relient l'espace joint à l'espace cartésien par la
relation
%        V=J(Theta)*Theta_prime).
%
%        [Tivim1,Riv0,Pid0,JVid0]=cindir(M,N,Alphaim1,Aim1,Di,Thetai,Ki)
%        retourne les matrices de transformation, les matrices de
rotation,
%        les vecteurs position et les Jacobiens de vitesse.
%
%        M=nombre d'articulations de la base au dernier moteur;
%        N=nombre total d'articulations (du référentiel 1 au référentiel
de
%          la pince);
%        Alphaim1=alpha_{i-1};
%        Aim1=a_{i-1};
%        Di=d_{i};
%        Thetai=theta_{i};
%        Ki indique un joint rotatif (Ki=1) ou prismatique (Ki=0).
%
%        Les variables des vecteur Thetai et Di sont
%        identifiées comme ceci: [Theta1; Theta2; ...] et [d1; d2; ...].
%
%        Les constantes Alphaim1 et Aim1 sont identifiées comme
%        ceci: [Alpha1; Alpha2; ...] et [L1; L2; ...].
%
%        Tivim1=vecteur des matrices de transformation reliant le
référentiel i
%               au référentiel i-1 (^{i-1}_{i}T) pour i=1 à M.
%        Riv0t= vecteur des matrices de rotation donnant l'orientation
du référentiel
%               i par rapport à la base (^{0}R_{i}) pour i=1 à M.
%        Pid0= vecteur des vecteurs de position du référentiel i par
rapport
%             à la base (^{0}P_{i}) pour i=1 à M.
%        JVid0= vecteur des Jacobiens de vitesse du référentiel i par
rapport à la
%               base (J_{i}) pour i=1 à M.

% Date de la dernière modification: 11/08/94
%                          03/04/98 : Theta -> T
%               20/05/98 : adaptation pour matlab 5.2
%               22/09/99 : 12 digits pour plus de précision
```

```
% Définition des principales variables employées :

% Tiv0 : matrice de transformation reliant le référentiel i à la base
% Zid0t : axe Z du référentiel i vu de la base
% Zid0XP0t :Si l'articulation est rotative, le vecteur de l'articulation
en question
%           est égal au produit vectoriel entre Zia0 et PNaid0 tandis
que si elle est
%           prismatique, le vecteur est égal à Zia0.
% Riv0t : matrice de rotation donnant l'orientation du référentiel i par
rapport à la base
% RNp1vi : vecteur des matrices de rotation du référentiel N+1
%          vers chaque référentiel
% Tivim1t : matrice de transformation reliant le référentiel i au
référentiel i-1
% PMdit : vecteur montrant la position du référentiel N par rapport au
référentiel i
% PMdi : vecteur des vecteurs PMdit
% Z : axe Z ( [0,0,1]' )
% Zid0 : vecteur des Zid0t pour tous les référentiels
% PMdid0 : vecteur PNai représenté dans la base
% Zid0XP0 : vecteur des Zid0XP0t pour tous les référentiels

digits(12)

% Définition des vecteurs de matrices symboliques

syms Tiviml Zid0 Riv0t Riv0 Pid0 Pid0t Pid0p b c d e Trans JVid0 real
Tiv0=eye(4);
Z=[0;0;1];
Zid0(1:3*N,1)=zeros(3*N,1);

% Formation des matrices de transformation
Trans=[cos(e),(-sin(e)),0,c;sin(e)*cos(b),cos(e)*cos(b),(-sin(b)),d*(-
sin(b));...
            sin(e)*sin(b),cos(e)*sin(b),cos(b),d*cos(b);0,0,0,1];

for I = 1:N

   i=N-I+1;

   Tivim1t = subs(Trans,'b',Alphaim1(i,1));
   Tivim1t = subs(Tivim1t,'c',Aim1(i,1));
   Tivim1t = subs(Tivim1t,'d',Di(i,1));
   Tivim1t = subs(Tivim1t,'e',Thetai(i,1));

   Tivim1=extract(Tivim1t,1,4,1,4,Tiviml,(i-1)*4+1,1);

end

for I=1:N
   % Prise de la matrice de transformation nécessaire dans le vecteur de
matrices Tivim1.

   Tivim1t=extrac(Tivim1,4*(I-1)+1,4*I,1,4);
```

```
   % Cumulation(multiplication) des matrices de transformation du
référentiel 0 au
   % référentiel a.
   Tiv0=Tiv0*Tivim1t;

   % Formation de la matrice de rotations cumulatives et du vecteur de
matrices de
   % rotation d'un référentiel par rapport au précédent.

   Riv0t=extrac(Tiv0,1,3,1,3);
   Riv0=extract(Tiv0,1,3,1,3,Riv0,1+(I-1)*3,1);
   Pid0=extract(Tiv0,1,3,4,4,Pid0,1+(I-1)*3,1);

   Zid0t=Riv0t*Z;
   % Formation du vecteur de matrices Z(Axe z d'un référentiel par
rapport au référentiel 1.
   % Prise du vecteur Pa(origine du référentiel n+1 par rapport au
référentiel a dans le
   % vecteur de vecteurs PNai.

   Zid0=extract(Zid0t,1,3,1,1,Zid0,3*(I-1)+1,1);

   % Formation des vecteurs de vitesse linéaire des Jacobiens de vitesse

   for J=1:min(I,M)

      Zid0t=extrac(Zid0,3*(J-1)+1,3*J,1,1);

      K=Ki(J,1);

      Pid0t=extrac(Pid0,(I-1)*3+1,(I-1)*3+3,1,1);

      Pid0tp=diff(Pid0t,(['T' num2str(J)]));

      Pid0tp = K*diff(Pid0t,(['T' num2str(J)])) + eval(['1-'
num2str(K)])*Zid0t;

      Pid0p=extract(Pid0tp,1,3,1,1,Pid0p,(I-1)*3+1,J);

   end

end

% Formation de la matrice jacobienne

for I=1:N
   for J=1:min(I,M)
      for L=1:3
         JVid0((I-1)*6+L,J)=Pid0p(3*(I-1)+L,J);
         if ((Ki(J,1)==1)&(abs(eval(Zid0((3*(J-1)+L),1)))>=1e-6))
            JVid0((I-1)*6+L+3,J)=Zid0((3*(J-1)+L),1);
         else
            JVid0((I-1)*6+L+3,J)=0;
         end
      end
   end
end
```

```
for a=1:N
   for b=1:M
      for c=1:6
         JVid0((a-1)*6+c,b)=vpa(simple(JVid0((a-1)*6+c,b)));
      end
   end
end

Tivim1=vpa(simplify(Tivim1));
Tivim1=vpa(simple(Tivim1));
Riv0=vpa(simple(Riv0));
Pid0=vpa(simple(Pid0));
```

Programme de lancement de CINDIR (produit automatiquement par le générateur de code) (cinrrr)

```
% CIN Calcule la cinématique directe pour x articulation(s).
%
% Le manipulateur est défini par la fonction defbrax fournie par
% l'utilisateur

%*%*%1 Appel de la fonction pour fichier de données.
[M,N,Alphaim1,Aim1,Di,Thetai,Ki]=defbrarrr;

% Calcul de la cinématique directe.
[Tivim1,Riv0,Pid0,JVid0]=cindir(M,N,Alphaim1,Aim1,Di,Thetai,Ki);

%*%*%2 Écriture du fichier de données de cinématique directe
save cinrrr.mat
```

Fichier de configuration de la géométrie du manipulateur nécessaire à l'exécution de CINDIR (defbrarrr)

```
function [M,N,Alphaim1,Aim1,Di,Thetai,Ki]=defbra;
% DEFBRA Définir les paramètres du manipulateur x pour la cinématique.
%
%       [M,N,Alphaim1,Aim1,Di,Thetai,Ki]=DEFBRA() retourne les
paramètres
%       du manipulateur selon la convention de Denavit-Hartenberg
modifiée.
%
%       M=nombre d'articulations de la base au dernier moteur;
%       N=nombre total d'articulations (du référentiel 1 au référentiel
de
%         la pince);
%       Alphaim1=alpha_{i-1};
%       Aim1=a_{i-1};
```

```
%        Di=d_{i};
%        Thetai=theta_{i};
%        Ki indique un joint rotatif (Ki=1) ou prismatique (Ki=0).
%
%        Les variables des vecteur Thetai et Di doivent être
%        identifiées comme ceci: [Theta1; Theta2; ...] et [d1; d2; ...].
%
%        Les constantes Alphaim1 et Aim1 doivent être identifiées comme
%        ceci: [Alpha1; Alpha2; ...] et [L1; L2; ...].
%

digits(12);

%Définition des variables symboliques
syms Alphaim1 Aim1 Di Thetai real

%*%*%1 initialisation de M, N, Alphaim1, Aim1, Di, Thetai, Ki
syms L0 L1 L2 L3 L4 T1 T2 T3 real

N=5
M=3
Alphaim1=[1.57079632680;0;0;0;0];
Aim1=[L0;L1;L2;L3;L4];
Di=[0;0;0;0;0];
Thetai=[T1;T2;T3;1.57079632680;0];
Ki=[1;1;1];
```

Programme DYNAM de calcul de la dynamique inverse d'un manipulateur rigide série à N joints (version adaptée au générateur de code sous la forme de Lagrange-Euler

```
function [Ti,MTheta,gTheta,VThThp] = dynam(fichier);
% DYNAM Trouve la dynamique inverse d'un manipulateur dont on a trouvé
la
%       cinématique directe avec CIN et qui a été redéfini dans DEFMAN
%

% Date de la dernière modification : 26/9/94
%                                    21/2/95 : ajout de "simple" et
compteur
%                             03/4/98 : Theta -> T
%                                       limite le nombre de digits au
début
%                                    retour des forces et couples pour
validation
%                22/05/98 : Conversion pour matlab 5.2
%                04/06/98 : Correction de bugs (initialisation des
%                           matrices utilisées)
%                20/2/99 : M devient N et N devient M
%                          M : Nombre d'articulations actives
```

```
%                          N : Nombre d'articulations total

% Nom des principales variables :
%
% qip : vecteur de variables de vitesse linéaire d'une articulation
% qipp : vecteur de variables d'accélération linéaire d'une
%        articulation
% Dp1, Dp2... : valeurs de qip pour chaque articulation prismatique
% Dpp1, Dpp2... : valeurs de qipp pour chaque articulation prismatique
% Fi : vecteur des forces agissant sur chaque membre
% Fit : valeur temporaire du vecteur Fi
% Icidi : matrice contenant tous les tenseurs d'inertie
% Masi : vecteur ligne définissant les tenseurs d'inertie de toutes
%        les masses
% Massei : matrice contenant la valeur de la masse de chaque masse
%          de chaque membre
% Ni : vecteur des couples agissant sur chaque articulation
% Nit : valeur temporaire de Ni
% PNp1di : vecteur des vecteurs position du référentiel N+1 par
%          rapport à chaque référentiel
% Pc : vecteur des position du centre de masse de chaque membre
%      par rapport au référentiel homologue au membre
% Pci : une des valeurs du vecteur Pc
% Pip1di : vecteur des vecteurs position de chaque référentiel
%          par rapport à son précédent
% Pip1dit : valeur d'un des vecteur position de Pip1di
% Rcivi : matrice des matrices de rotation de chaque tenseur d'inertie
% Rip1vi : vecteur des matrices de rotation de chaque référentiel
%          vers son précédent
% Rip1vit : une des matrices de rotation du vecteur de matrices
%           Rip1vi
% Rivim1t : une des matrices de rotation du vecteur de matrices
%           de transformation Tivim1
% Rivip1 : vecteur des matrices de rotation de chaque référentiel
%          vers le référentiel suivant
% Rivip1t : une des matrices de rotation du vecteur de matrices
%           Rivip1
% Ti : vecteur des couples aux joints
% Tit : un des couple du vecteur Ti
% Tp1, Tp2... : valeurs du vecteur qip pour des articulations
%                       rotatives
% Tpp1, Tpp2... : valeurs du vecteur qipp pour des articulations
%                         rotatives
% Tivim1 : vecteur des matrices de transformation de chaque
%          référentiel par rapport au précédent
% qp : valeur temporaire d'une des valeurs de qip
% qpp : valeur temporaire d'une des valeurs de qipp
% axeX : vecteur [1,0,0]'
% axeZ : vecteur [0,0,1]'
% ccmdh : vecteur ligne contenant les coordonnées des tenseurs d'inertie
%         par rapport à l'articulation contenant la masse en notation
%         Denavit-Hartenberg modifiée
% ffi : vecteur des vecteurs de force exercées sur chaque lien par
%       le lien précédent
% fip1 : une valeur du vecteur ffi
% fit : une valeur du vecteur ffi
% m : vecteur de masse de chaque membre
```

```
% mi : valeur temporaire du vecteur m
% ni : vecteur des vecteurs de couple exercés sur chaque lien par
%      le lien précédent
% nip1 : une des valeurs de ni
% nit : une des valeurs de ni
% vcp : vecteur des accélérations linéaires des centres de masse
%       de chaque membre
% vcpi : une des accélérations du vecteur vcp
% vp : vecteur des accélérations linéaires de l'origine de chaque
%      référentiel des membres par rapport à leur référentiel homologue
% vpi : une des valeurs du vecteur vp
% vpip1 : une des valeurs du vecteur vp
% w : vecteur des vitesses angulaires de chaque référentiel des
%     membres par rapport à leur référentiel homologue
% wi : vecteur des accélérations angulaires de chaque référentiel des
%      membres par rapport à leur référentiel homologue
% wiXP : produit vectoriel d'un des éléments du vecteur wi par le
%        vecteur position Pip1dit correspondant
% wiXPci : produit vectoriel d'un des éléments du vecteur wi par le
%          vecteur position Pci correspondant
% wip1 : une des valeurs de wi
% wp : vecteur des accélérations angulaires de chaque référentiel
%      des membres par rapport au référentiel homologue
% wpi : une des valeurs du vecteur wp
% wpip1 : une des valeurs du vecteur wp

% Limite le nombre de décimales après la virgule
digits(9);

eval(['[Tivim1,Ki,M,Mi,Pc,Rcivi,ccmdh,Masi,Massei,Icidi,w,wp,vp,ffi,ni]=
' fichier ';']);

syms Rivim1t Rivip1 Pip1di Rivip1t Rip1vi SFi SNi Rcivit real
syms vcp Fit Nit Fi Ni Ti wi wpi vpi qip qipp qp qpp wip1 wpip1 vpip1
real
syms wiXP vcpi Pci wiXPci mi Icidit real
syms MTheta VThThp gTheta real

% Initialisation des matrices de rotations et des vecteurs porteurs de
% matrices de rotation
Rivim1t(1:3,1:3)=zeros(3);
Rivip1(1:3*(M+1),1:3)=zeros(3*(M+1),3);
Rivip1t(1:3,1:3)=zeros(3);
Rip1vi(1:3*(M+1),1:3)=zeros(3*(M+1),3);
Rcivit(1:3,1:3)=zeros(3);

% Initialisation des vecteurs et des vecteurs porteurs de vecteur
Pip1di(1:3*(M+1),1)=zeros(3*(M+1),1);
vcp(1:3*M,1)=zeros(3*M,1);
Fit(1:3,1)=zeros(3,1);
Nit(1:3,1)=zeros(3,1);
Fi(1:3*M,1)=zeros(3*M,1);
Ni(1:3*M,1)=zeros(3*M,1);
Ti(1:M,1)=zeros(M,1);
wi(1:3,1)=zeros(3,1);
wpi(1:3,1)=zeros(3,1);
vpi(1:3,1)=zeros(3,1);
```

```
qip(1:M,1)=zeros(M,1);
qipp(1:M,1)=zeros(M,1);
wip1(1:3,1)=zeros(3,1);
wpip1(1:3,1)=zeros(3,1);
vpip1(1:3,1)=zeros(3,1);
wiXP(1:3,1)=zeros(3,1);
vcpi(1:3,1)=zeros(3,1);
Pci(1:3,1)=zeros(3,1);
wiXPci(1:3,1)=zeros(3,1);
Icidit(1:3,1:3)=zeros(3);

% Initialisation des matrices et vecteurs de dynamique directe
MTheta(1:M,1:M)=zeros(M);
gTheta(1:M,1)=zeros(M,1);
VThThp(1:M,1)=zeros(M,1);

for L=1:M+1
   Rivim1t=extrac(Tivim1,(L-1)*4+1,(L-1)*4+3,1,3);

   Rip1vi=extract(Tivim1,(L-1)*4+1,(L-1)*4+3,1,3,Rip1vi,(L-1)*3+1,1);

   Rivip1t=Rivim1t.';

   Rivip1=extract(Rivip1t,1,3,1,3,Rivip1,(L-1)*3+1,1);

end

for I=1:M+1
   Pip1di=extract(Tivim1,(I-1)*4+1,(I-1)*4+3,4,4,Pip1di,(I-1)*3+1,1);
end

% Calcul des vitesses et accélérations

for I=1:M
%disp(['cal v et a: I= ' num2str(I)])

   wi=extrac(w,(I-1)*3+1,(I-1)*3+3,1,1);
   wpi=extrac(wp,(I-1)*3+1,(I-1)*3+3,1,1);
   vpi=extrac(vp,(I-1)*3+1,(I-1)*3+3,1,1);

   Rivip1t=extrac(Rivip1,(I-1)*3+1,(I-1)*3+3,1,3);
   Pip1dit=extrac(Tivim1,(I-1)*4+1,(I-1)*4+3,4,4);

   % Composition des variables articulaires du moment présent (i+1)
   K=Ki(I,1);
   if K==1
      qip(I,1)=['Tp' num2str(I)];
      qipp(I,1)=['Tpp' num2str(I)];
      qp=qip(I,1);
      qpp=qipp(I,1);
   else
      qip(I,1)=['Dp' num2str(I)];
      qipp(I,1)=['Dpp' num2str(I)];
      qp=qip(I,1);
      qpp=qipp(I,1);
   end
```

```
axeZ=[0;0;1];

wip1 = Rivip1t*wi + qp*axeZ*K;
wip1 = simple(wip1);

w = extract(wip1,1,3,1,1,w,I*3+1,1);

wpip1 = Rivip1t*wpi + (cross((Rivip1t*wi),(qp*axeZ)) + qpp*axeZ)*K;
wpip1=simple(wpip1);

wp=extract(wpip1,1,3,1,1,wp,I*3+1,1);

wiXP=cross(wi,Pip1dit);

vpip1 = Rivip1t*(cross(wpi,Pip1dit) + cross(wi,wiXP) + vpi) + ...
   (cross((2*wip1),(qp*axeZ)) + qpp*axeZ)*eval(['1-' num2str(K)]);
vpip1=simple(vpip1);

vp=extract(vpip1,1,3,1,1,vp,I*3+1,1);

% Construction des vecteurs d'accélérations linéaires des centres
% de masse pour chaque masse et pour chaque membre.

for J=1:Mi(I,1)
   %disp(['cons v et a: J= ' num2str(J)])

   Pci=extrac(Pc,(I-1)*3+1,(I-1)*3+3,J,J);

   wiXPci=cross(wip1,Pci);

   vcpi = cross(wpip1,Pci) + cross(wip1,wiXPci) + vpip1;

   vcp=extract(vcpi,1,3,1,1,vcp,I*3+1,J);

end

vcp=simple(vcp);

% Calcul des forces et couples agissant sur les membres en appliquant
% les équations de Newton-Euler (pour chaque masse).

for J=1:Mi(I,1)
   %disp(['cal ff et c: J= ' num2str(J)])
   mi=Massei(I,J);
   vcpi=extrac(vcp,I*3+1,I*3+3,J,J);
   wi=extrac(w,I*3+1,I*3+3,1,1);
   wpi=extrac(wp,I*3+1,I*3+3,1,1);
   Icidit=extrac(Icidi,(I-1)*3+1,(I-1)*3+3,(J-1)*3+1,(J-1)*3+3);
   Rcivit=extrac(Rcivi,(I-1)*3+1,(I-1)*3+3,(J-1)*3+1,(J-1)*3+3);

   % Multiplication de la matrice d'inertie par la matrice
   % de rotation pour représenter correctement l'orientation
   % de la masse par rapport au membre.

   Icidit = Rcivit.'*(Icidit*Rcivit);

   Icidit=simple(Icidit);
```

```
      Fit = mi*vcpi;

      Nit = Icidit*wpi + cross(wi,(Icidit*wi));

      Fi=extract(Fit,1,3,1,1,Fi,(I-1)*3+1,J);
      Ni=extract(Nit,1,3,1,1,Ni,(I-1)*3+1,J);

   end

   Fi=simple(Fi);
   Ni=simple(Ni);

   % Formation des vecteurs SFi et SNi représentant la sommation des
   % vecteurs de forces et de couples pour chaque membre.

   SFit(1:3,1)=zeros(3,1);
   SNit(1:3,1)=zeros(3,1);
   for J=1:Mi(I,1)

      Fit=extrac(Fi,(I-1)*3+1,(I-1)*3+3,J,J);
      SFit=SFit+Fit;
      Nit=extrac(Ni,(I-1)*3+1,(I-1)*3+3,J,J);
      SNit=SNit+Nit;

   end

   SFi=extract(SFit,1,3,1,1,SFi,(I-1)*3+1,1);
   SNi=extract(SNit,1,3,1,1,SNi,(I-1)*3+1,1);

end

   SFi=simple(SFi);
   SNi=simple(SNi);

% Itérations vers la base pour calculer les forces et couples

for I=1:M
   J=M-I+1;
   %disp(['iter cal ff et c: I= ' num2str(I)])
   Rip1vit=extrac(Rip1vi,J*3+1,J*3+3,1,3);
   fip1=extrac(ffi,J*3+1,J*3+3,1,1);
   Fit=extrac(SFi,(J-1)*3+1,(J-1)*3+3,1,1);
   nip1=extrac(ni,J*3+1,J*3+3,1,1);
   Nit=extrac(SNi,(J-1)*3+1,(J-1)*3+3,1,1);
   Pip1dit=extrac(Pip1di,J*3+1,J*3+3,1,1);
   Kit=extrac(Ki,J,J,1,1);

   fit = Rip1vit*fip1 + Fit;
   fit=simple(fit);

   ffi=extract(fit,1,3,1,1,ffi,(J-1)*3+1,1);

   SPciXFit(1:3,1)=zeros(3,1);
   for L=1:Mi(J,1)

      Pci=extrac(Pc,(J-1)*3+1,(J-1)*3+3,L,L);
```

```
      Fit=extrac(Fi,(J-1)*3+1,(J-1)*3+3,L,L);
      PciXFit = cross(Pci,Fit);
      SPciXFit = SPciXFit + PciXFit;

   end

   SPciXFit=simple(SPciXFit);

   nit = Nit + Rip1vit*nip1 + SPciXFit + cross(Pip1dit,(Rip1vit*fip1));
   nit = simple(nit);
   ni=extract(nit,1,3,1,1,ni,(J-1)*3+1,1);

   if Kit=='1'
      Tit = nit.'*axeZ;
   end

   if Kit=='0'
      Tit = fit.'*axeZ;
   end

   Ti=extract(Tit,1,1,1,1,Ti,J,1);

end

Ti=simplify(Ti);
Ti=simple(Ti);

%disp(['debut factorisation de Ti'])

% Formation des matrices de masse, du vecteur de forces et couples dûs à
la
% gravité et du vecteur de forces et couples provenant des effets
centrifuges
% et Coriolis.

for I=1:M
   for J=1:M

      MTheta(I,J) = diff(Ti(I,1),qipp(J,1));

   end
end
MTheta=simplify(MTheta);
MTheta=vpa(simple(MTheta));

Tit=Ti;
for I=1:M

   Tit=subs(Tit,sym(['Tpp' num2str(I)]),0);
   Tit=subs(Tit,sym(['Tp' num2str(I)]),0);
   Tit=subs(Tit,sym(['Dpp' num2str(I)]),0);
   Tit=subs(Tit,sym(['Dp' num2str(I)]),0);

end

gTheta = Tit;
gTheta=simplify(gTheta);
```

```
gTheta=vpa(simple(gTheta));

VThThp = Ti - MTheta*qipp - gTheta;
VThThp=simplify(VThThp);
VThThp=vpa(simple(VThThp));

% Production de la matrice de masse inverse pour pouvoir trouver
% la dynamique directe

%MThetainv=inv(MTheta);
%MThetainv=simple(MThetainv);
```

Fichier de configuration des masses du manipulateur nécessaire à l'exécution de DYNAM (defmanrrr)

```
function
[Tivim1,Ki,M,Mi,Pc,Rcivi,ccmdh,Masi,Massei,Icidi,w,wp,vp,ffi,ni]=defman;
% DEFMAN est un complément de DEFBRA qui fournit à DYN les paramètres
%        nécessaires à la résolution de la dynamique inverse.
%
% Par : Martin de Montigny, Pierre Sicard
% Dernière modification : 2 février 1999
%
% Note: Les données à changer d'un manipulateur à l'autre sont précédés
de %*%*%

%*%*%1 Chargement du fichier de sauvegarde des données de la cinématique
directe
load cinrrr.mat;

%*%*%2 Définition du vecteur du nombre de masses par membre
Mi = [1;1;1];

%%%%% Coordonnées des centres de masse avec notation Denavit-Hartenberg
modifiée
% On doit entrer les données sur une seule rangée comme ci-dessous :
% ccmdh = sym(['[Alpha_{i-1}masse1membre1,A_{i-1},D_{i},Theta_{i},' ...
%               'Alpha_{i-1}masse2membre1,A_{i-1},D_{i},Theta_{i},' ...
%               'Alpha_{i-1}masse1membre2,A_{i-1},D_{i},Theta_{i}]']);
%
%*%*%3 (définition des lx et de ccmdh)
syms l0 l1 l2 real
ccmdh = [0,l0,0,0,...
         0,l1,0,0,...
         0,l2,0,0];

% Formation des vecteurs de position des centres de masse et des
matrices de
% rotation donnant l'orientation de chaque masse par rapport à
l'articulation.

[Rcivi,Pc]=POSCEN(M,Mi,ccmdh);
```

```
%%%%% Définition de la matrice de paramètres des masses donnant le type
de masse,
% la masse totale et les paramètres du type de masse.  Pour chaque
masse, on
% doit définir les paramètres de la masse comme dans l'exemple ci-
dessous :
% Masi = sym(['[nom,masse totale,paramètre1,paramètre2,0,0,0,0,0,0,0,
...
%               nom,masse totale,0,0,0,0,0,0,0,0,0; ...
%               nom,masse
totale,paramètre1,paramètre2,paramètre3,0,0,0,0,0,0]']);
%
% On doit noter que chaque ligne contient les paramètres de toutes les
masses
% de l'articulation du même rang et que chaque colonne contient les
données
% de la première masse de chaque membre.
%
% Les noms et paramètres requis pour ajouter une masse d'un certain type
est
% décrit ci-dessous:
%
% type                        nom par1  par2  par3  par4 ... par9
%
% masse-point               : MPO 0     0     0     0    ... 0
% cylindre plein uniforme   : CPU h     r     0     0    ... 0
% cylindre creux uniforme   : CCU h     r     0     0    ... 0
% cylindre mince uniforme   : CMU h     0     0     0    ... 0
% prisme                    : PRI a     b     c     0    ... 0
% définie par utilisateur   : DPU Ixx   -Ixy  -Ixz  -Iyx {*}
%
% {*} : Iyy, -Iyz, -Izx, -Izy, Izz
%
%*%*%4 (Définition des masses et éléments des tenseurs de même que Masi)
syms DPU M0 M1 M2 real
Masi = [DPU,M0,0,0,0,0,0,0,0,0,0,...
        DPU,M1,0,0,0,0,0,0,0,0,0,...
        DPU,M2,0,0,0,0,0,0,0,0,0];

% Formation des matrices d'inertie à partir des données
% de l'utilisateur

[Icidi,Massei]=tenseurs(Mi,M,Masi);

%%%%% Conditions initiales de la base (ne tourne pas + effets de la
gravité)

w=zeros((M+1)*3,1);
w=sym(w,'d');
wp=zeros((M+1)*3,1);
wp=sym(wp,'d');

%%%%% Définir l'effet du vecteur de gravité sur le référentiel zero
% vp correspond à l'accélération requise de la base pour simuler l'effet
de
% la gravité.
syms g vp real
```

```
%*%*%5 (vecteur opposé à la gravité)
vp(1,1)=vpa(0);
vp(2,1)=vpa(0);
vp(3,1)=vpa(9.81);

%%%%% Inclusion de l'effet d'un contact avec l'environnement (force et
couple)

ffi = zeros((M+1)*3,1);
ffi = sym(ffi,'d');
ni = zeros((M+1)*3,1);
ni = sym(ni,'d');
```

Sous routines nécessaires à l'exécution de CINDIR et DYNAM

Routine d'extraction d'un vecteur à partir d'un vecteur maître (extrac)

```
function [destination]=extrac(source,ligmin,ligmax,colmin,colmax)
% EXTRAC Extraction d'une partie de la matrice 'source' symbolique.
%
%       [destination]=extrac(source,ligmin,ligmax,colmin,colmax)
%       extrait une partie de la matrice 'source' délimitée par les
bornes
%       ligmin (ligne minimale), ligmax (ligne maximale), colmin
(colonne
%       minimale) et colmax (colonne maximale), et enregistre ces
éléments
%       dans la matrice 'destination'.

% Date de la dernière modification : 20/05/1998

syms destination real

for I=ligmin:ligmax
   for J=colmin:colmax
      destination(I-ligmin+1,J-colmin+1)=source(I,J);
   end
end
```

Routine d'extraction (version 2) d'un vecteur à partir d'un vecteur maître (extract)

```
function [destination]=extract(source,ligmin,ligmax,colmin,colmax, ...
                               destination,ligdep,coldep)
```

```
% EXTRACT Extraction d'une partie de la matrice 'source' symbolique.
%
%         [destination]=extract(source,ligmin,ligmax,colmin,colmax, ...
%                               destination,ligdep,coldep)
%         extrait une partie de la matrice 'source' délimitée par les
bornes
%         ligmin (ligne minimale), ligmax (ligne maximale), colmin
(colonne
%         minimale) et colmax (colonne maximale), et enregistre ces
éléments
%         dans la matrice 'destination' à partir de la coordonnée ligdep
%         (ligne de départ) et coldep (colonne de départ).

% Date de la dernière modification : 20/05/1998

for I=ligmin:ligmax
   for J=colmin:colmax
      destination(I-ligmin+ligdep,J-colmin+coldep)=source(I,J);
   end
end
```

Routine de définition de l'emplacement et de l'orientation des masses associées à chaque membre (poscen)

```
function [Rcivi,Pc]=poscen(M,Mi,ccmdh);
% POSCEN Construction des matrices de rotation et des vecteurs de
position reliant
%        le référentiel attaché à chacune des masses au référentiel du
membre associé.
%
%     [Rcivi,Pc]=poscen(M,Mi,ccmdh) extrait les paramètres établis par
la
%       convention Denavit-Hartenberg modifiée pour définir la position
et
%     l'orientation du référentiel attaché à chacune des masses par
rapport
%     au référentiel du membre associé.
%
%     M= nombre de membres actifs;
%     Mi= vecteur avec N éléments représentant le nombre de masses
fixées à
%       chacun des membres;
%     ccmdh= matrice de somme(Mi)X4 contenant les paramètres D-H pour
chacun des
%         référentiel de masse:
%         ccmdh = [Alpha_{i-1}masse1membre1,A_{i-1},D_{i},Theta_{i},
%                  Alpha_{i-1}masse2membre1,A_{i-1},D_{i},Theta_{i},
%                ...
%                  Alpha_{i-1}masse1membre2,A_{i-1},D_{i},Theta_{i},
%                ...
%                Alpha_{i-1}masseMi(N)membreN,A_{i-1},D_{i},Theta_{i}]
%       Les paramètres Alpha_{i-1} et Theta_{i} (avec A_{i-1} = D_{i} =
0)
```

```
%       définissent la rotation nécessaire pour aligner les référentiels
de la
%       charge et du membre;
%       les paramètres A_{i-1} et D_{i} (avec Alpha_{i-1} = Theta_{i} =
0) définissent
%       la translation nécessaire (après la rotation) pour faire
coincider l'origine
%       des référentiels.  La transformation s'effectue donc en
employant un référentiel
%       intermédiaire.
%


% Formation des matrices de transformation
% Les 4 premières lignes sont consacrées au membre 1, les lignes 5 à 8,
au membre 2...
% Les 4 premières colonnes représentent les premières masses de chaque
membre...


syms Pc Rcivi b c d e real

Rota=[cos(e),-sin(e),0;sin(e)*cos(b),cos(e)*cos(b),-
sin(b);sin(e)*sin(b), ...
          cos(e)*sin(b),cos(b)];
Posi=[c;0;d];

compteur=0;
for I = 1:M

   for J = 1:Mi(I,1)

      Rcivit = subs(Rota,'b',ccmdh(1,(J-1+compteur)*4+1));
      Rcivit = subs(Rcivit,'e',ccmdh(1,(J-1+compteur)*4+4));
      Pct = subs(Posi,'c',ccmdh(1,(J-1+compteur)*4+2));
      Pct = subs(Pct,'d',ccmdh(1,(J-1+compteur)*4+3));

      Rcivi=extract(Rcivit,1,3,1,3,Rcivi,(I-1)*3+1,(J-1)*3+1);
      Pc=extract(Pct,1,3,1,1,Pc,(I-1)*3+1,J);

   end
   compteur=compteur+Mi(I,1);
end

Rcivi=simplify(Rcivi);
Pc=simplify(Pc);
```

Routine de fabrication des tenseurs d'inertie pour chaque masse du maniopulateur

```
function [Icidi,Massei]=tenseurs(Mi,M,Masi);
% Tenseurs fabrique les tenseurs d'inertie pour chaque masse de chaque
%          articulation à partir des données contenues dans le vecteur
```

```
%          ligne Masi.

syms Ixx Iyy Izz Massei Icidi real

compteur=0;
for I=1:M
   for J=1:Mi(I,1)

      m = Masi(1,(J-1+compteur)*11+2);
      Massei(I,J) = m;

      Icidit = [Ixx,0,0;0,Iyy,0;0,0,Izz];

      if Masi(1,(J-1+compteur)*11+1) == 'MPO'

         Icidit = [0,0,0;0,0,0;0,0,0];

      elseif Masi(1,(J-1+compteur)*11+1) == 'CPU'

         h=Masi(1,(J-1+compteur)*11+3);
         r=Masi(1,(J-1+compteur)*11+4);

         Icidit = subs(Icidit,Ixx,m*(3*r^2+h^2)/12);
         Icidit = subs(Icidit,Iyy,m*(3*r^2+h^2)/12);
         Icidit = subs(Icidit,Izz,m*r^2/2);

      elseif Masi(1,(J-1+compteur)*11+1) == 'CCU'
         h=Masi(1,(J-1+compteur)*11+3);
         r=Masi(1,(J-1+compteur)*11+4);

         Icidit=subs(Icidit,Ixx,m*(6*r^2+h^2)/6);
         Icidit=subs(Icidit,Iyy,m*(6*r^2+r^2)/6);
         Icidit=subs(Icidit,Izz,m*r^2);


      elseif Masi(1,(J-1+compteur)*11+1) == 'CMU'
         h=Masi(1,(J-1+compteur)*11+3);

         Icidit=subs(Icidit,Ixx,m*h^2/12);
         Icidit=subs(Icidit,Iyy,m*h^2/12);
         Icidit=subs(Icidit,Izz,0);

      elseif Masi(1,(J-1+compteur)*11+1) == 'PRI'
         a=Masi(1,(J-1+compteur)*11+3);
         b=Masi(1,(J-1+compteur)*11+4);
         c=Masi(1,(J-1+compteur)*11+5);

         Icidit=subs(Icidit,Ixx,m*(b^2+c^2)/12);
         Icidit=subs(Icidit,Iyy,m*(a^2+c^2)/12);
         Icidit=subs(Icidit,Izz,m*(a^2+b^2)/12);

      elseif Masi(1,(J-1+compteur)*11+1) == 'DPU'

         for L=1:3
            for P=1:3
               Icidit(L,P)=Masi(1,(J-1+compteur)*11+2+3*(L-1)+P);
            end
```

```
         end

      end

      Icidi=extract(Icidit,1,3,1,3,Icidi,(I-1)*3+1,(J-1)*3+1);

   end
   compteur=compteur+Mi(I,1);
end
```

Annexe E

Cinématique inverse du manipulateur étudié

La figure E.1 présente la structure des référentiels d'un des doigts :

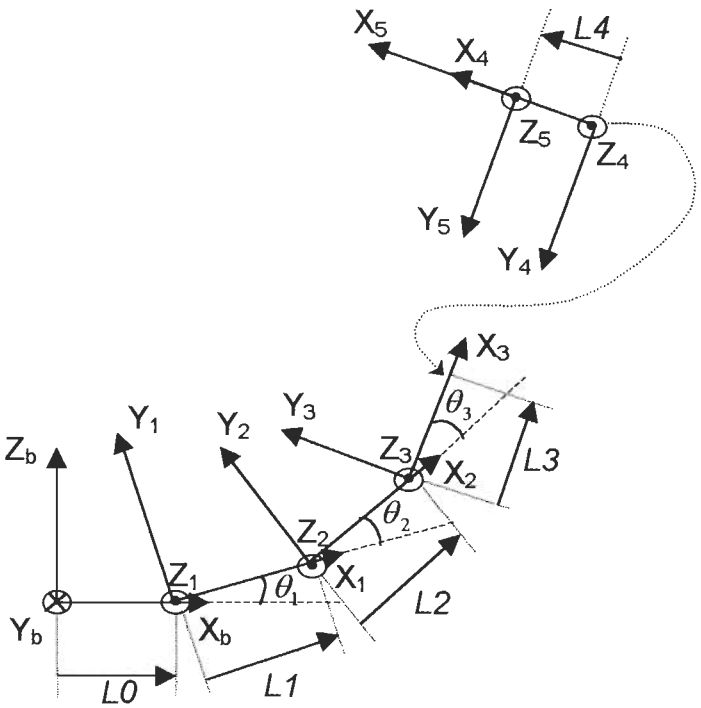

Figure E. 1 : Structure géométrique d'un doigt de la main modélisée

Cette structure de manipulateur possède deux configurations possibles. La première, que nous n'utiliserons pas, et pour laquelle l'angle θ_2 est négatif produirait possiblement des collisions non modélisées entre les doigts (superposition des membres de différents doigts). Nous choisissons donc le cas pour lequel θ_2 est positif.

Pour calculer la cinématique inverse, nous procédons en plusieurs étapes. Premièrement, voyons à la figure E.2 un schéma d'un doigt du premier au troisième référentiel :

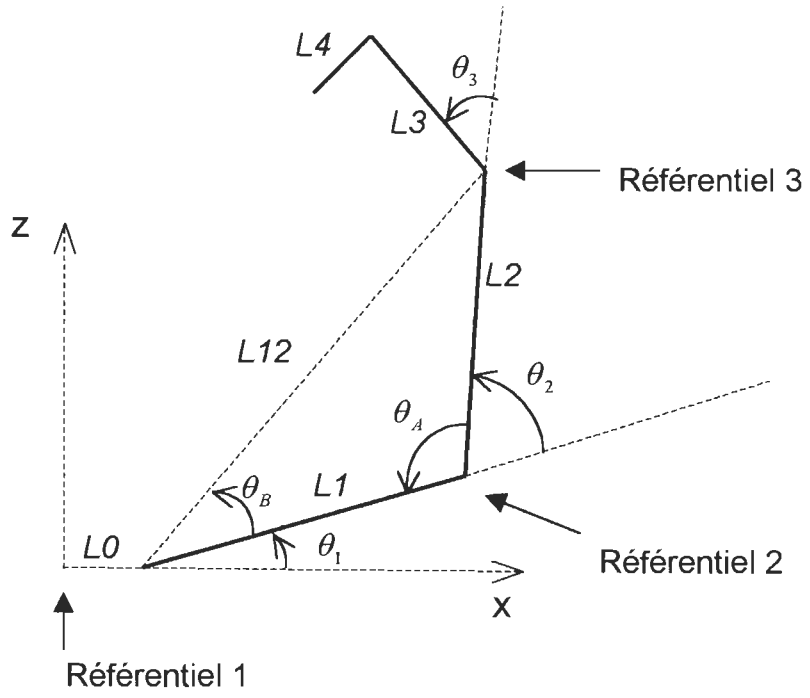

Figure E. 2 : Schéma permettant de calculer la cinématique inverse

Pour débuter, nous calculons les positions en x et en z de l'origine du référentiel 3.

$$\begin{aligned} p4x &= p5x + L4\sin(\theta_1+\theta_2+\theta_3) \\ p4z &= p5z - L4\cos(\theta_1+\theta_2+\theta_3) \\ p3x &= p4x - L3\cos(\theta_1+\theta_2+\theta_3) \\ p3z &= p4z - L3\sin(\theta_1+\theta_2+\theta_3) \end{aligned} \tag{E.1}$$

Nous savons que la position du référentiel 1 est toujours nulle en z et égal à $L0$ en x. Nous pouvons ensuite calculer la longueur $L12$ avec la formule :

$$L12 = \sqrt{(p3x - L0)^2 + (p3z - 0)^2} \tag{E.2}$$

Ensuite, par la loi des cosinus, nous obtenons θ_A :

$$\theta_A = \arccos\left(\frac{L2^2 + L1^2 - L12^2}{2L1L2}\right) \tag{E.3}$$

Nous pouvons maintenant facilement obtenir θ_2 :

$$\theta_2 = \pi - \theta_A \tag{E.4}$$

Par la suite, nous obtenons θ_B par (E.5) car la somme des angles d'un triangle est de π :

$$\theta_B = \frac{\pi - \theta_A}{2} = \frac{\theta_2}{2} \tag{E.5}$$

Ensuite, nous calculons l'angle de la droite (p3z-p1z,p3x-p1x) :

$$\theta_{Droite} = \arctan\left(\frac{p3z - p1z}{p3x - p1x}\right) = \arctan\left(\frac{p3z}{p3x - L0}\right) \tag{E.6}$$

Finalement, nous calculons θ_1 :

$$\theta_1 = \theta_{Droite} - \theta_B \tag{E.7}$$

$$\theta_3 = \theta_{123} - \theta_1 - \theta_2 \tag{E.8}$$

où θ_{123} caractérise l'orientation de l'effecteur (l'orientation de l'effecteur $\theta_e = \theta_{123} + \pi/2$ sur la figure E.2).

Annexe F

Fonctions d'animation du manipulateur

Fonction d'animation en 3 dimensions du manipulateur en temps réel sous la forme d'une s-function de Matlab

```
function [sys, x0, str, ts] = sfun3d(t,x,u,flag)
% SFUN3D.M
%
% Cette fonction effectue une animation en 3 dimensions d'une main
% à 3 doigts.
%
% Par : Martin de Montigny
%       Pierre Sicard
% UQTR
% Opal-rt
% Juillet 1998

switch flag

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0
    [sys,x0,ts] = mdlInitializeSizes;

  %%%%%%%%%%
  % Update %
  %%%%%%%%%%
  case 2
    sys = mdlUpdate(t,x,u,flag);

  %%%%%%%%%%%%%%%%
  % Unused flags %
  %%%%%%%%%%%%%%%%
  case { 3, 9 }
    sys = [];

end

% end sfun3d

%
%=============================================================================
======
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
function.
%=============================================================================
======
%
function [sys,x0,ts] = mdlInitializeSizes

sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 0;
```

```
sizes.NumInputs      = 14;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

x0 = [];

ts=[-1 0]; %période d'échantillonnage adaptée au bloc précédent

% end mdlInitializeSizes

%
%=============================================================================
======
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
======
%
function sys=mdlUpdate(t,x,u,flag)

%
% always return empty, there are no states...
%
sys = [];

%
% Locate the figure window associated with this block.  If it's not a
valid
% handle (it may have been closed by the user), then return.
%
FigHandle=GetSfunXYFigure(gcbh);

if ~ishandle(FigHandle),
    return
end

%
% Get UserData of the figure.
%
ud = get(FigHandle,'UserData');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if t==0

 %
 % the figure doesn't exist, create one
 %

 FigHandle = visu3;

 elevt=30;
 azimt=30;
 set(FigHandle,'renderer','zbuffer');
```

```
 ha=get(FigHandle,'children');
 %set(ha(6),'projection','perspective');%mettre un tag.....(ralentit
de 20%...)
 set(ha(6),'drawmode','fast');
 set(ha(6),'Visible','off');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
 %%%%%%%   Début du calcul des points du tracé 3d
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

 % matrices de rotation des bases 1,2 et 3
 rb1=eye(3);
 rb2=[cos(2*pi/3),-sin(2*pi/3),0;sin(2*pi/3),cos(2*pi/3),0;0,0,1];
 rb3=[cos(-2*pi/3),-sin(-2*pi/3),0;sin(-2*pi/3),cos(-2*pi/3),0;0,0,1];

 % Calcul des points de la base
 larg=(u(11)+u(12)+u(13)+u(14))/4*0.8;
 offset=larg*tan(pi/6)/10; % Offset de la base en x pour permettre la
construction de
                                                    % l'illusion d'une
seule base...
 ptsb1=[larg/20+u(10),-larg/10,larg/20;larg/20+u(10),-larg/10,-larg/20;
...
        u(10),-larg/10,-larg/10;offset,-larg/10,-larg/10; ...
        offset,-larg/10,larg/10;u(10),-larg/10,larg/10; ...
        larg/20+u(10),larg/10,larg/20;larg/20+u(10),larg/10,-larg/20;
...
        u(10),larg/10,-larg/10;offset,larg/10,-larg/10; ...
        offset,larg/10,larg/10;u(10),larg/10,larg/10];

 % Calcul des points des membres
 ptsm1=[-larg/20,larg/20,-larg*0.07;0,larg/10,-larg*0.07; ...
       u(11)-larg/5,larg/10,-larg/10;u(11),larg/10,-larg/10; ...
       u(11)+larg/20,larg/20,-larg/10;u(11)+larg/20,-larg/20,-larg/10;
...
       u(11),-larg/10,-larg/10;u(11)-larg/5,-larg/10,-larg/10; ...
       0,-larg/10,-larg*0.07;-larg/20,-larg/20,-larg*0.07; ...
       -larg/20,larg/20,larg*0.07;0,larg/10,larg*0.07; ...
       u(11)-larg/5,larg/10,larg/10;u(11),larg/10,larg/10; ...
       u(11)+larg/20,larg/20,larg/10;u(11)+larg/20,-larg/20,larg/10; ...
       u(11),-larg/10,larg/10;u(11)-larg/5,-larg/10,larg/10; ...
       0,-larg/10,larg*0.07;-larg/20,-larg/20,larg*0.07];
 ptsm2=[-larg/20,larg/20,-larg*0.07;0,larg/10,-larg*0.07; ...
       u(12)-larg/5,larg/10,-larg/10;u(12),larg/10,-larg/10; ...
       u(12)+larg/20,larg/20,-larg/10;u(12)+larg/20,-larg/20,-larg/10;
...
       u(12),-larg/10,-larg/10;u(12)-larg/5,-larg/10,-larg/10; ...
       0,-larg/10,-larg*0.07;-larg/20,-larg/20,-larg*0.07; ...
       -larg/20,larg/20,larg*0.07;0,larg/10,larg*0.07; ...
       u(12)-larg/5,larg/10,larg/10;u(12),larg/10,larg/10; ...
       u(12)+larg/20,larg/20,larg/10;u(12)+larg/20,-larg/20,larg/10; ...
       u(12),-larg/10,larg/10;u(12)-larg/5,-larg/10,larg/10; ...
```

```
      0,-larg/10,larg*0.07;-larg/20,-larg/20,larg*0.07];
 ptsm3=[-larg/20,larg/20,-larg*0.07;0,larg/10,-larg*0.07; ...
      u(13)-larg/5,larg/10,-larg/10;u(13),larg/10,-larg/10; ...
      u(13)+larg/20,larg/20,-larg/10;u(13)+larg/20,-larg/20,-larg/10;
...
      u(13),-larg/10,-larg/10;u(13)-larg/5,-larg/10,-larg/10; ...
      0,-larg/10,-larg*0.07;-larg/20,-larg/20,-larg*0.07; ...
      -larg/20,larg/20,larg*0.07;0,larg/10,larg*0.07; ...
      u(13)-larg/5,larg/10,larg/10;u(13),larg/10,larg/10; ...
      u(13)+larg/20,larg/20,larg/10;u(13)+larg/20,-larg/20,larg/10; ...
      u(13),-larg/10,larg/10;u(13)-larg/5,-larg/10,larg/10; ...
      0,-larg/10,larg*0.07;-larg/20,-larg/20,larg*0.07];

 %calcul des points de l'effecteur
 ptse1=[-larg/20,larg/20,-larg*0.07;0,larg/10,-larg*0.07; ...
      u(14)-larg/20,larg/10,-larg*0.07;u(14),larg/20,-larg/20; ...
      u(14),-larg/20,-larg/20;u(14)-larg/20,-larg/10,-larg*0.07; ...
      0,-larg/10,-larg*0.07;-larg/20,-larg/20,-larg*0.07; ...
      -larg/20,larg/20,larg*0.07;0,larg/10,larg*0.07; ...
      u(14)-larg/20,larg/10,larg*0.07;u(14),larg/20,larg/20; ...
      u(14),-larg/20,larg/20;u(14)-larg/20,-larg/10,larg*0.07; ...
      0,-larg/10,larg*0.07;-larg/20,-larg/20,larg*0.07];

 % Met le vecteur de points dans le bon sens pour la multiplication avec
les
 % lignes de la matrice de rotation
 ptsb1=ptsb1';
 ptsb2=[rb2(1,:)*ptsb1;rb2(2,:)*ptsb1;rb2(3,:)*ptsb1];
 ptsb3=[rb3(1,:)*ptsb1;rb3(2,:)*ptsb1;rb3(3,:)*ptsb1];

 ptsm1_1=ptsm1';
 ptsm1_2=ptsm2';
 ptsm1_3=ptsm3';
 ptse1=ptse1';

 % Formation des matrices de position des points des formes définissant
la base 1
 xr=[ptsb1(1,1),ptsb1(1,6),ptsb1(1,11),ptsb1(1,12),ptsb1(1,7); ...
    ptsb1(1,2),ptsb1(1,3),ptsb1(1,10),ptsb1(1,9),ptsb1(1,8); ...
    ptsb1(1,3),ptsb1(1,4),ptsb1(1,4),ptsb1(1,10),ptsb1(1,9); ...
    ptsb1(1,6),ptsb1(1,5),ptsb1(1,5),ptsb1(1,11),ptsb1(1,12)];

 yr=[ptsb1(2,1),ptsb1(2,6),ptsb1(2,11),ptsb1(2,12),ptsb1(2,7); ...
     ptsb1(2,2),ptsb1(2,3),ptsb1(2,10),ptsb1(2,9),ptsb1(2,8); ...
     ptsb1(2,3),ptsb1(2,4),ptsb1(2,4),ptsb1(2,10),ptsb1(2,9); ...
     ptsb1(2,6),ptsb1(2,5),ptsb1(2,5),ptsb1(2,11),ptsb1(2,12)];

 zr=[ptsb1(3,1),ptsb1(3,6),ptsb1(3,11),ptsb1(3,12),ptsb1(3,7); ...
     ptsb1(3,2),ptsb1(3,3),ptsb1(3,10),ptsb1(3,9),ptsb1(3,8); ...
     ptsb1(3,3),ptsb1(3,4),ptsb1(3,4),ptsb1(3,10),ptsb1(3,9); ...
     ptsb1(3,6),ptsb1(3,5),ptsb1(3,5),ptsb1(3,11),ptsb1(3,12)];

 cb=zeros(4,5); %couleur de la base

 % Formation des matrices de position des points des formes définissant
le membre 1(doigt1)
```

```
xr=[xr,[ptsm1_1(1,1),ptsm1_1(1,2),ptsm1_1(1,3),ptsm1_1(1,4), ...
   ptsm1_1(1,1),ptsm1_1(1,2),ptsm1_1(1,10),ptsm1_1(1,9), ...
   ptsm1_1(1,11),ptsm1_1(1,12),ptsm1_1(1,13),ptsm1_1(1,14); ...
   ptsm1_1(1,2),ptsm1_1(1,3),ptsm1_1(1,4),ptsm1_1(1,5), ...
   ptsm1_1(1,2),ptsm1_1(1,3),ptsm1_1(1,9),ptsm1_1(1,8), ...
   ptsm1_1(1,12),ptsm1_1(1,13),ptsm1_1(1,14),ptsm1_1(1,15); ...
   ptsm1_1(1,9),ptsm1_1(1,8),ptsm1_1(1,7),ptsm1_1(1,6), ...
   ptsm1_1(1,12),ptsm1_1(1,13),ptsm1_1(1,19),ptsm1_1(1,18), ...
   ptsm1_1(1,19),ptsm1_1(1,18),ptsm1_1(1,17),ptsm1_1(1,16); ...
   ptsm1_1(1,10),ptsm1_1(1,9),ptsm1_1(1,8),ptsm1_1(1,7), ...
   ptsm1_1(1,11),ptsm1_1(1,12),ptsm1_1(1,20),ptsm1_1(1,19), ...
   ptsm1_1(1,20),ptsm1_1(1,19),ptsm1_1(1,18),ptsm1_1(1,17)]];

yr=[yr,[ptsm1_1(2,1),ptsm1_1(2,2),ptsm1_1(2,3),ptsm1_1(2,4), ...
   ptsm1_1(2,1),ptsm1_1(2,2),ptsm1_1(2,10),ptsm1_1(2,9), ...
   ptsm1_1(2,11),ptsm1_1(2,12),ptsm1_1(2,13),ptsm1_1(2,14); ...
   ptsm1_1(2,2),ptsm1_1(2,3),ptsm1_1(2,4),ptsm1_1(2,5), ...
   ptsm1_1(2,2),ptsm1_1(2,3),ptsm1_1(2,9),ptsm1_1(2,8), ...
   ptsm1_1(2,12),ptsm1_1(2,13),ptsm1_1(2,14),ptsm1_1(2,15); ...
   ptsm1_1(2,9),ptsm1_1(2,8),ptsm1_1(2,7),ptsm1_1(2,6), ...
   ptsm1_1(2,12),ptsm1_1(2,13),ptsm1_1(2,19),ptsm1_1(2,18), ...
   ptsm1_1(2,19),ptsm1_1(2,18),ptsm1_1(2,17),ptsm1_1(2,16); ...
   ptsm1_1(2,10),ptsm1_1(2,9),ptsm1_1(2,8),ptsm1_1(2,7), ...
   ptsm1_1(2,11),ptsm1_1(2,12),ptsm1_1(2,20),ptsm1_1(2,19), ...
   ptsm1_1(2,20),ptsm1_1(2,19),ptsm1_1(2,18),ptsm1_1(2,17)]];

zr=[zr,[ptsm1_1(3,1),ptsm1_1(3,2),ptsm1_1(3,3),ptsm1_1(3,4), ...
   ptsm1_1(3,1),ptsm1_1(3,2),ptsm1_1(3,10),ptsm1_1(3,9), ...
   ptsm1_1(3,11),ptsm1_1(3,12),ptsm1_1(3,13),ptsm1_1(3,14); ...
   ptsm1_1(3,2),ptsm1_1(3,3),ptsm1_1(3,4),ptsm1_1(3,5), ...
   ptsm1_1(3,2),ptsm1_1(3,3),ptsm1_1(3,9),ptsm1_1(3,8), ...
   ptsm1_1(3,12),ptsm1_1(3,13),ptsm1_1(3,14),ptsm1_1(3,15); ...
   ptsm1_1(3,9),ptsm1_1(3,8),ptsm1_1(3,7),ptsm1_1(3,6), ...
   ptsm1_1(3,12),ptsm1_1(3,13),ptsm1_1(3,19),ptsm1_1(3,18), ...
   ptsm1_1(3,19),ptsm1_1(3,18),ptsm1_1(3,17),ptsm1_1(3,16); ...
   ptsm1_1(3,10),ptsm1_1(3,9),ptsm1_1(3,8),ptsm1_1(3,7), ...
   ptsm1_1(3,11),ptsm1_1(3,12),ptsm1_1(3,20),ptsm1_1(3,19), ...
   ptsm1_1(3,20),ptsm1_1(3,19),ptsm1_1(3,18),ptsm1_1(3,17)]];

cm1=0.25*ones(4,12);

% Formation des matrices de position des points des formes définissant
le membre 2(doigt1)

xr=[xr,[ptsm1_2(1,1),ptsm1_2(1,2),ptsm1_2(1,3),ptsm1_2(1,4), ...
   ptsm1_2(1,1),ptsm1_2(1,2),ptsm1_2(1,10),ptsm1_2(1,9), ...
   ptsm1_2(1,11),ptsm1_2(1,12),ptsm1_2(1,13),ptsm1_2(1,14); ...
   ptsm1_2(1,2),ptsm1_2(1,3),ptsm1_2(1,4),ptsm1_2(1,5), ...
   ptsm1_2(1,2),ptsm1_2(1,3),ptsm1_2(1,9),ptsm1_2(1,8), ...
   ptsm1_2(1,12),ptsm1_2(1,13),ptsm1_2(1,14),ptsm1_2(1,15); ...
   ptsm1_2(1,9),ptsm1_2(1,8),ptsm1_2(1,7),ptsm1_2(1,6), ...
   ptsm1_2(1,12),ptsm1_2(1,13),ptsm1_2(1,19),ptsm1_2(1,18), ...
   ptsm1_2(1,19),ptsm1_2(1,18),ptsm1_2(1,17),ptsm1_2(1,16); ...
   ptsm1_2(1,10),ptsm1_2(1,9),ptsm1_2(1,8),ptsm1_2(1,7), ...
   ptsm1_2(1,11),ptsm1_2(1,12),ptsm1_2(1,20),ptsm1_2(1,19), ...
   ptsm1_2(1,20),ptsm1_2(1,19),ptsm1_2(1,18),ptsm1_2(1,17)]];
```

```
yr=[yr,[ptsm1_2(2,1),ptsm1_2(2,2),ptsm1_2(2,3),ptsm1_2(2,4), ...
   ptsm1_2(2,1),ptsm1_2(2,2),ptsm1_2(2,10),ptsm1_2(2,9), ...
   ptsm1_2(2,11),ptsm1_2(2,12),ptsm1_2(2,13),ptsm1_2(2,14); ...
   ptsm1_2(2,2),ptsm1_2(2,3),ptsm1_2(2,4),ptsm1_2(2,5), ...
   ptsm1_2(2,2),ptsm1_2(2,3),ptsm1_2(2,9),ptsm1_2(2,8), ...
   ptsm1_2(2,12),ptsm1_2(2,13),ptsm1_2(2,14),ptsm1_2(2,15); ...
   ptsm1_2(2,9),ptsm1_2(2,8),ptsm1_2(2,7),ptsm1_2(2,6), ...
   ptsm1_2(2,12),ptsm1_2(2,13),ptsm1_2(2,19),ptsm1_2(2,18), ...
   ptsm1_2(2,19),ptsm1_2(2,18),ptsm1_2(2,17),ptsm1_2(2,16); ...
   ptsm1_2(2,10),ptsm1_2(2,9),ptsm1_2(2,8),ptsm1_2(2,7), ...
   ptsm1_2(2,11),ptsm1_2(2,12),ptsm1_2(2,20),ptsm1_2(2,19), ...
   ptsm1_2(2,20),ptsm1_2(2,19),ptsm1_2(2,18),ptsm1_2(2,17)]];

zr=[zr,[ptsm1_2(3,1),ptsm1_2(3,2),ptsm1_2(3,3),ptsm1_2(3,4), ...
   ptsm1_2(3,1),ptsm1_2(3,2),ptsm1_2(3,10),ptsm1_2(3,9), ...
   ptsm1_2(3,11),ptsm1_2(3,12),ptsm1_2(3,13),ptsm1_2(3,14); ...
   ptsm1_2(3,2),ptsm1_2(3,3),ptsm1_2(3,4),ptsm1_2(3,5), ...
   ptsm1_2(3,2),ptsm1_2(3,3),ptsm1_2(3,9),ptsm1_2(3,8), ...
   ptsm1_2(3,12),ptsm1_2(3,13),ptsm1_2(3,14),ptsm1_2(3,15); ...
   ptsm1_2(3,9),ptsm1_2(3,8),ptsm1_2(3,7),ptsm1_2(3,6), ...
   ptsm1_2(3,12),ptsm1_2(3,13),ptsm1_2(3,19),ptsm1_2(3,18), ...
   ptsm1_2(3,19),ptsm1_2(3,18),ptsm1_2(3,17),ptsm1_2(3,16); ...
   ptsm1_2(3,10),ptsm1_2(3,9),ptsm1_2(3,8),ptsm1_2(3,7), ...
   ptsm1_2(3,11),ptsm1_2(3,12),ptsm1_2(3,20),ptsm1_2(3,19), ...
   ptsm1_2(3,20),ptsm1_2(3,19),ptsm1_2(3,18),ptsm1_2(3,17)]];

cm2=0.5*ones(4,12);


% Formation des matrices de position des points des formes définissant
le membre 3(doigt1)

xr=[xr,[ptsm1_3(1,1),ptsm1_3(1,2),ptsm1_3(1,3),ptsm1_3(1,4), ...
   ptsm1_3(1,1),ptsm1_3(1,2),ptsm1_3(1,10),ptsm1_3(1,9), ...
   ptsm1_3(1,11),ptsm1_3(1,12),ptsm1_3(1,13),ptsm1_3(1,14); ...
   ptsm1_3(1,2),ptsm1_3(1,3),ptsm1_3(1,4),ptsm1_3(1,5), ...
   ptsm1_3(1,2),ptsm1_3(1,3),ptsm1_3(1,9),ptsm1_3(1,8), ...
   ptsm1_3(1,12),ptsm1_3(1,13),ptsm1_3(1,14),ptsm1_3(1,15); ...
   ptsm1_3(1,9),ptsm1_3(1,8),ptsm1_3(1,7),ptsm1_3(1,6), ...
   ptsm1_3(1,12),ptsm1_3(1,13),ptsm1_3(1,19),ptsm1_3(1,18), ...
   ptsm1_3(1,19),ptsm1_3(1,18),ptsm1_3(1,17),ptsm1_3(1,16); ...
   ptsm1_3(1,10),ptsm1_3(1,9),ptsm1_3(1,8),ptsm1_3(1,7), ...
   ptsm1_3(1,11),ptsm1_3(1,12),ptsm1_3(1,20),ptsm1_3(1,19), ...
   ptsm1_3(1,20),ptsm1_3(1,19),ptsm1_3(1,18),ptsm1_3(1,17)]];

yr=[yr,[ptsm1_3(2,1),ptsm1_3(2,2),ptsm1_3(2,3),ptsm1_3(2,4), ...
   ptsm1_3(2,1),ptsm1_3(2,2),ptsm1_3(2,10),ptsm1_3(2,9), ...
   ptsm1_3(2,11),ptsm1_3(2,12),ptsm1_3(2,13),ptsm1_3(2,14); ...
   ptsm1_3(2,2),ptsm1_3(2,3),ptsm1_3(2,4),ptsm1_3(2,5), ...
   ptsm1_3(2,2),ptsm1_3(2,3),ptsm1_3(2,9),ptsm1_3(2,8), ...
   ptsm1_3(2,12),ptsm1_3(2,13),ptsm1_3(2,14),ptsm1_3(2,15); ...
   ptsm1_3(2,9),ptsm1_3(2,8),ptsm1_3(2,7),ptsm1_3(2,6), ...
   ptsm1_3(2,12),ptsm1_3(2,13),ptsm1_3(2,19),ptsm1_3(2,18), ...
   ptsm1_3(2,19),ptsm1_3(2,18),ptsm1_3(2,17),ptsm1_3(2,16); ...
   ptsm1_3(2,10),ptsm1_3(2,9),ptsm1_3(2,8),ptsm1_3(2,7), ...
   ptsm1_3(2,11),ptsm1_3(2,12),ptsm1_3(2,20),ptsm1_3(2,19), ...
   ptsm1_3(2,20),ptsm1_3(2,19),ptsm1_3(2,18),ptsm1_3(2,17)]];
```

```
zr=[zr,[ptsm1_3(3,1),ptsm1_3(3,2),ptsm1_3(3,3),ptsm1_3(3,4), ...
   ptsm1_3(3,1),ptsm1_3(3,2),ptsm1_3(3,10),ptsm1_3(3,9), ...
   ptsm1_3(3,11),ptsm1_3(3,12),ptsm1_3(3,13),ptsm1_3(3,14); ...
   ptsm1_3(3,2),ptsm1_3(3,3),ptsm1_3(3,4),ptsm1_3(3,5), ...
   ptsm1_3(3,2),ptsm1_3(3,3),ptsm1_3(3,9),ptsm1_3(3,8), ...
   ptsm1_3(3,12),ptsm1_3(3,13),ptsm1_3(3,14),ptsm1_3(3,15); ...
   ptsm1_3(3,9),ptsm1_3(3,8),ptsm1_3(3,7),ptsm1_3(3,6), ...
   ptsm1_3(3,12),ptsm1_3(3,13),ptsm1_3(3,19),ptsm1_3(3,18), ...
   ptsm1_3(3,19),ptsm1_3(3,18),ptsm1_3(3,17),ptsm1_3(3,16); ...
   ptsm1_3(3,10),ptsm1_3(3,9),ptsm1_3(3,8),ptsm1_3(3,7), ...
   ptsm1_3(3,11),ptsm1_3(3,12),ptsm1_3(3,20),ptsm1_3(3,19), ...
   ptsm1_3(3,20),ptsm1_3(3,19),ptsm1_3(3,18),ptsm1_3(3,17)]];

cm3=0.75*ones(4,12);


% Formation des matrices de position des points des formes définissant
l'effecteur 1

xr=[xr,[ptse1(1,1),ptse1(1,2),ptse1(1,3),ptse1(1,1), ...
        ptse1(1,2),ptse1(1,3),ptse1(1,4),ptse1(1,8), ...
        ptse1(1,7),ptse1(1,6),ptse1(1,9),ptse1(1,10), ...
        ptse1(1,11); ...
        ptse1(1,2),ptse1(1,3),ptse1(1,4),ptse1(1,2), ...
        ptse1(1,3),ptse1(1,4),ptse1(1,5),ptse1(1,7), ...
        ptse1(1,6),ptse1(1,5),ptse1(1,10),ptse1(1,11), ...
        ptse1(1,12); ...
        ptse1(1,7),ptse1(1,6),ptse1(1,5),ptse1(1,10), ...
        ptse1(1,11),ptse1(1,12),ptse1(1,13),ptse1(1,15), ...
        ptse1(1,14),ptse1(1,13),ptse1(1,15),ptse1(1,14), ...
        ptse1(1,13); ...
        ptse1(1,8),ptse1(1,7),ptse1(1,6),ptse1(1,9), ...
        ptse1(1,10),ptse1(1,11),ptse1(1,12),ptse1(1,16), ...
        ptse1(1,15),ptse1(1,14),ptse1(1,16),ptse1(1,15), ...
        ptse1(1,14)]];

yr=[yr,[ptse1(2,1),ptse1(2,2),ptse1(2,3),ptse1(2,1), ...
        ptse1(2,2),ptse1(2,3),ptse1(2,4),ptse1(2,8), ...
        ptse1(2,7),ptse1(2,6),ptse1(2,9),ptse1(2,10), ...
        ptse1(2,11); ...
        ptse1(2,2),ptse1(2,3),ptse1(2,4),ptse1(2,2), ...
        ptse1(2,3),ptse1(2,4),ptse1(2,5),ptse1(2,7), ...
        ptse1(2,6),ptse1(2,5),ptse1(2,10),ptse1(2,11), ...
        ptse1(2,12); ...
        ptse1(2,7),ptse1(2,6),ptse1(2,5),ptse1(2,10), ...
        ptse1(2,11),ptse1(2,12),ptse1(2,13),ptse1(2,15), ...
        ptse1(2,14),ptse1(2,13),ptse1(2,15),ptse1(2,14), ...
        ptse1(2,13); ...
        ptse1(2,8),ptse1(2,7),ptse1(2,6),ptse1(2,9), ...
        ptse1(2,10),ptse1(2,11),ptse1(2,12),ptse1(2,16), ...
        ptse1(2,15),ptse1(2,14),ptse1(2,16),ptse1(2,15), ...
        ptse1(2,14)]];

zr=[zr,[ptse1(3,1),ptse1(3,2),ptse1(3,3),ptse1(3,1), ...
        ptse1(3,2),ptse1(3,3),ptse1(3,4),ptse1(3,8), ...
        ptse1(3,7),ptse1(3,6),ptse1(3,9),ptse1(3,10), ...
```

```
        ptse1(3,11); ...
        ptse1(3,2),ptse1(3,3),ptse1(3,4),ptse1(3,2), ...
        ptse1(3,3),ptse1(3,4),ptse1(3,5),ptse1(3,7), ...
        ptse1(3,6),ptse1(3,5),ptse1(3,10),ptse1(3,11), ...
        ptse1(3,12); ...
        ptse1(3,7),ptse1(3,6),ptse1(3,5),ptse1(3,10), ...
        ptse1(3,11),ptse1(3,12),ptse1(3,13),ptse1(3,15), ...
        ptse1(3,14),ptse1(3,13),ptse1(3,15),ptse1(3,14), ...
        ptse1(3,13); ...
        ptse1(3,8),ptse1(3,7),ptse1(3,6),ptse1(3,9), ...
        ptse1(3,10),ptse1(3,11),ptse1(3,12),ptse1(3,16), ...
        ptse1(3,15),ptse1(3,14),ptse1(3,16),ptse1(3,15), ...
        ptse1(3,14)]];

ce=1*ones(4,13);

 % Formation des matrices de position des points des formes définissant
la base 2
 xr=[xr,[ptsb2(1,1),ptsb2(1,6),ptsb2(1,11),ptsb2(1,12),ptsb2(1,7); ...
     ptsb2(1,2),ptsb2(1,3),ptsb2(1,10),ptsb2(1,9),ptsb2(1,8); ...
     ptsb2(1,3),ptsb2(1,4),ptsb2(1,4),ptsb2(1,10),ptsb2(1,9); ...
     ptsb2(1,6),ptsb2(1,5),ptsb2(1,5),ptsb2(1,11),ptsb2(1,12)]];

 yr=[yr,[ptsb2(2,1),ptsb2(2,6),ptsb2(2,11),ptsb2(2,12),ptsb2(2,7); ...
     ptsb2(2,2),ptsb2(2,3),ptsb2(2,10),ptsb2(2,9),ptsb2(2,8); ...
     ptsb2(2,3),ptsb2(2,4),ptsb2(2,4),ptsb2(2,10),ptsb2(2,9); ...
     ptsb2(2,6),ptsb2(2,5),ptsb2(2,5),ptsb2(2,11),ptsb2(2,12)]];

 zr=[zr,[ptsb2(3,1),ptsb2(3,6),ptsb2(3,11),ptsb2(3,12),ptsb2(3,7); ...
     ptsb2(3,2),ptsb2(3,3),ptsb2(3,10),ptsb2(3,9),ptsb2(3,8); ...
     ptsb2(3,3),ptsb2(3,4),ptsb2(3,4),ptsb2(3,10),ptsb2(3,9); ...
     ptsb2(3,6),ptsb2(3,5),ptsb2(3,5),ptsb2(3,11),ptsb2(3,12)]];

 % Formation des matrices de position des points des formes définissant
le membre 1(doigt2)

 xr=[xr,[ptsm1_1(1,1),ptsm1_1(1,2),ptsm1_1(1,3),ptsm1_1(1,4), ...
    ptsm1_1(1,1),ptsm1_1(1,2),ptsm1_1(1,10),ptsm1_1(1,9), ...
    ptsm1_1(1,11),ptsm1_1(1,12),ptsm1_1(1,13),ptsm1_1(1,14); ...
    ptsm1_1(1,2),ptsm1_1(1,3),ptsm1_1(1,4),ptsm1_1(1,5), ...
    ptsm1_1(1,2),ptsm1_1(1,3),ptsm1_1(1,9),ptsm1_1(1,8), ...
    ptsm1_1(1,12),ptsm1_1(1,13),ptsm1_1(1,14),ptsm1_1(1,15); ...
    ptsm1_1(1,9),ptsm1_1(1,8),ptsm1_1(1,7),ptsm1_1(1,6), ...
    ptsm1_1(1,12),ptsm1_1(1,13),ptsm1_1(1,19),ptsm1_1(1,18), ...
    ptsm1_1(1,19),ptsm1_1(1,18),ptsm1_1(1,17),ptsm1_1(1,16); ...
    ptsm1_1(1,10),ptsm1_1(1,9),ptsm1_1(1,8),ptsm1_1(1,7), ...
    ptsm1_1(1,11),ptsm1_1(1,12),ptsm1_1(1,20),ptsm1_1(1,19), ...
    ptsm1_1(1,20),ptsm1_1(1,19),ptsm1_1(1,18),ptsm1_1(1,17)]];

 yr=[yr,[ptsm1_1(2,1),ptsm1_1(2,2),ptsm1_1(2,3),ptsm1_1(2,4), ...
    ptsm1_1(2,1),ptsm1_1(2,2),ptsm1_1(2,10),ptsm1_1(2,9), ...
    ptsm1_1(2,11),ptsm1_1(2,12),ptsm1_1(2,13),ptsm1_1(2,14); ...
    ptsm1_1(2,2),ptsm1_1(2,3),ptsm1_1(2,4),ptsm1_1(2,5), ...
    ptsm1_1(2,2),ptsm1_1(2,3),ptsm1_1(2,9),ptsm1_1(2,8), ...
    ptsm1_1(2,12),ptsm1_1(2,13),ptsm1_1(2,14),ptsm1_1(2,15); ...
    ptsm1_1(2,9),ptsm1_1(2,8),ptsm1_1(2,7),ptsm1_1(2,6), ...
    ptsm1_1(2,12),ptsm1_1(2,13),ptsm1_1(2,19),ptsm1_1(2,18), ...
```

```
   ptsm1_1(2,19),ptsm1_1(2,18),ptsm1_1(2,17),ptsm1_1(2,16); ...
   ptsm1_1(2,10),ptsm1_1(2,9),ptsm1_1(2,8),ptsm1_1(2,7), ...
   ptsm1_1(2,11),ptsm1_1(2,12),ptsm1_1(2,20),ptsm1_1(2,19), ...
   ptsm1_1(2,20),ptsm1_1(2,19),ptsm1_1(2,18),ptsm1_1(2,17)]];

zr=[zr,[ptsm1_1(3,1),ptsm1_1(3,2),ptsm1_1(3,3),ptsm1_1(3,4), ...
   ptsm1_1(3,1),ptsm1_1(3,2),ptsm1_1(3,10),ptsm1_1(3,9), ...
   ptsm1_1(3,11),ptsm1_1(3,12),ptsm1_1(3,13),ptsm1_1(3,14); ...
   ptsm1_1(3,2),ptsm1_1(3,3),ptsm1_1(3,4),ptsm1_1(3,5), ...
   ptsm1_1(3,2),ptsm1_1(3,3),ptsm1_1(3,9),ptsm1_1(3,8), ...
   ptsm1_1(3,12),ptsm1_1(3,13),ptsm1_1(3,14),ptsm1_1(3,15); ...
   ptsm1_1(3,9),ptsm1_1(3,8),ptsm1_1(3,7),ptsm1_1(3,6), ...
   ptsm1_1(3,12),ptsm1_1(3,13),ptsm1_1(3,19),ptsm1_1(3,18), ...
   ptsm1_1(3,19),ptsm1_1(3,18),ptsm1_1(3,17),ptsm1_1(3,16); ...
   ptsm1_1(3,10),ptsm1_1(3,9),ptsm1_1(3,8),ptsm1_1(3,7), ...
   ptsm1_1(3,11),ptsm1_1(3,12),ptsm1_1(3,20),ptsm1_1(3,19), ...
   ptsm1_1(3,20),ptsm1_1(3,19),ptsm1_1(3,18),ptsm1_1(3,17)]];

 % Formation des matrices de position des points des formes définissant
le membre 2(doigt2)

xr=[xr,[ptsm1_2(1,1),ptsm1_2(1,2),ptsm1_2(1,3),ptsm1_2(1,4), ...
   ptsm1_2(1,1),ptsm1_2(1,2),ptsm1_2(1,10),ptsm1_2(1,9), ...
   ptsm1_2(1,11),ptsm1_2(1,12),ptsm1_2(1,13),ptsm1_2(1,14); ...
   ptsm1_2(1,2),ptsm1_2(1,3),ptsm1_2(1,4),ptsm1_2(1,5), ...
   ptsm1_2(1,2),ptsm1_2(1,3),ptsm1_2(1,9),ptsm1_2(1,8), ...
   ptsm1_2(1,12),ptsm1_2(1,13),ptsm1_2(1,14),ptsm1_2(1,15); ...
   ptsm1_2(1,9),ptsm1_2(1,8),ptsm1_2(1,7),ptsm1_2(1,6), ...
   ptsm1_2(1,12),ptsm1_2(1,13),ptsm1_2(1,19),ptsm1_2(1,18), ...
   ptsm1_2(1,19),ptsm1_2(1,18),ptsm1_2(1,17),ptsm1_2(1,16); ...
   ptsm1_2(1,10),ptsm1_2(1,9),ptsm1_2(1,8),ptsm1_2(1,7), ...
   ptsm1_2(1,11),ptsm1_2(1,12),ptsm1_2(1,20),ptsm1_2(1,19), ...
   ptsm1_2(1,20),ptsm1_2(1,19),ptsm1_2(1,18),ptsm1_2(1,17)]];

yr=[yr,[ptsm1_2(2,1),ptsm1_2(2,2),ptsm1_2(2,3),ptsm1_2(2,4), ...
   ptsm1_2(2,1),ptsm1_2(2,2),ptsm1_2(2,10),ptsm1_2(2,9), ...
   ptsm1_2(2,11),ptsm1_2(2,12),ptsm1_2(2,13),ptsm1_2(2,14); ...
   ptsm1_2(2,2),ptsm1_2(2,3),ptsm1_2(2,4),ptsm1_2(2,5), ...
   ptsm1_2(2,2),ptsm1_2(2,3),ptsm1_2(2,9),ptsm1_2(2,8), ...
   ptsm1_2(2,12),ptsm1_2(2,13),ptsm1_2(2,14),ptsm1_2(2,15); ...
   ptsm1_2(2,9),ptsm1_2(2,8),ptsm1_2(2,7),ptsm1_2(2,6), ...
   ptsm1_2(2,12),ptsm1_2(2,13),ptsm1_2(2,19),ptsm1_2(2,18), ...
   ptsm1_2(2,19),ptsm1_2(2,18),ptsm1_2(2,17),ptsm1_2(2,16); ...
   ptsm1_2(2,10),ptsm1_2(2,9),ptsm1_2(2,8),ptsm1_2(2,7), ...
   ptsm1_2(2,11),ptsm1_2(2,12),ptsm1_2(2,20),ptsm1_2(2,19), ...
   ptsm1_2(2,20),ptsm1_2(2,19),ptsm1_2(2,18),ptsm1_2(2,17)]];

zr=[zr,[ptsm1_2(3,1),ptsm1_2(3,2),ptsm1_2(3,3),ptsm1_2(3,4), ...
   ptsm1_2(3,1),ptsm1_2(3,2),ptsm1_2(3,10),ptsm1_2(3,9), ...
   ptsm1_2(3,11),ptsm1_2(3,12),ptsm1_2(3,13),ptsm1_2(3,14); ...
   ptsm1_2(3,2),ptsm1_2(3,3),ptsm1_2(3,4),ptsm1_2(3,5), ...
   ptsm1_2(3,2),ptsm1_2(3,3),ptsm1_2(3,9),ptsm1_2(3,8), ...
   ptsm1_2(3,12),ptsm1_2(3,13),ptsm1_2(3,14),ptsm1_2(3,15); ...
   ptsm1_2(3,9),ptsm1_2(3,8),ptsm1_2(3,7),ptsm1_2(3,6), ...
   ptsm1_2(3,12),ptsm1_2(3,13),ptsm1_2(3,19),ptsm1_2(3,18), ...
   ptsm1_2(3,19),ptsm1_2(3,18),ptsm1_2(3,17),ptsm1_2(3,16); ...
   ptsm1_2(3,10),ptsm1_2(3,9),ptsm1_2(3,8),ptsm1_2(3,7), ...
```

```
   ptsm1_2(3,11),ptsm1_2(3,12),ptsm1_2(3,20),ptsm1_2(3,19), ...
   ptsm1_2(3,20),ptsm1_2(3,19),ptsm1_2(3,18),ptsm1_2(3,17)]];

 % Formation des matrices de position des points des formes définissant
le membre 3(doigt2)

 xr=[xr,[ptsm1_3(1,1),ptsm1_3(1,2),ptsm1_3(1,3),ptsm1_3(1,4), ...
    ptsm1_3(1,1),ptsm1_3(1,2),ptsm1_3(1,10),ptsm1_3(1,9), ...
    ptsm1_3(1,11),ptsm1_3(1,12),ptsm1_3(1,13),ptsm1_3(1,14); ...
    ptsm1_3(1,2),ptsm1_3(1,3),ptsm1_3(1,4),ptsm1_3(1,5), ...
    ptsm1_3(1,2),ptsm1_3(1,3),ptsm1_3(1,9),ptsm1_3(1,8), ...
    ptsm1_3(1,12),ptsm1_3(1,13),ptsm1_3(1,14),ptsm1_3(1,15); ...
    ptsm1_3(1,9),ptsm1_3(1,8),ptsm1_3(1,7),ptsm1_3(1,6), ...
    ptsm1_3(1,12),ptsm1_3(1,13),ptsm1_3(1,19),ptsm1_3(1,18), ...
    ptsm1_3(1,19),ptsm1_3(1,18),ptsm1_3(1,17),ptsm1_3(1,16); ...
    ptsm1_3(1,10),ptsm1_3(1,9),ptsm1_3(1,8),ptsm1_3(1,7), ...
    ptsm1_3(1,11),ptsm1_3(1,12),ptsm1_3(1,20),ptsm1_3(1,19), ...
    ptsm1_3(1,20),ptsm1_3(1,19),ptsm1_3(1,18),ptsm1_3(1,17)]];

 yr=[yr,[ptsm1_3(2,1),ptsm1_3(2,2),ptsm1_3(2,3),ptsm1_3(2,4), ...
    ptsm1_3(2,1),ptsm1_3(2,2),ptsm1_3(2,10),ptsm1_3(2,9), ...
    ptsm1_3(2,11),ptsm1_3(2,12),ptsm1_3(2,13),ptsm1_3(2,14); ...
    ptsm1_3(2,2),ptsm1_3(2,3),ptsm1_3(2,4),ptsm1_3(2,5), ...
    ptsm1_3(2,2),ptsm1_3(2,3),ptsm1_3(2,9),ptsm1_3(2,8), ...
    ptsm1_3(2,12),ptsm1_3(2,13),ptsm1_3(2,14),ptsm1_3(2,15); ...
    ptsm1_3(2,9),ptsm1_3(2,8),ptsm1_3(2,7),ptsm1_3(2,6), ...
    ptsm1_3(2,12),ptsm1_3(2,13),ptsm1_3(2,19),ptsm1_3(2,18), ...
    ptsm1_3(2,19),ptsm1_3(2,18),ptsm1_3(2,17),ptsm1_3(2,16); ...
    ptsm1_3(2,10),ptsm1_3(2,9),ptsm1_3(2,8),ptsm1_3(2,7), ...
    ptsm1_3(2,11),ptsm1_3(2,12),ptsm1_3(2,20),ptsm1_3(2,19), ...
    ptsm1_3(2,20),ptsm1_3(2,19),ptsm1_3(2,18),ptsm1_3(2,17)]];

 zr=[zr,[ptsm1_3(3,1),ptsm1_3(3,2),ptsm1_3(3,3),ptsm1_3(3,4), ...
    ptsm1_3(3,1),ptsm1_3(3,2),ptsm1_3(3,10),ptsm1_3(3,9), ...
    ptsm1_3(3,11),ptsm1_3(3,12),ptsm1_3(3,13),ptsm1_3(3,14); ...
    ptsm1_3(3,2),ptsm1_3(3,3),ptsm1_3(3,4),ptsm1_3(3,5), ...
    ptsm1_3(3,2),ptsm1_3(3,3),ptsm1_3(3,9),ptsm1_3(3,8), ...
    ptsm1_3(3,12),ptsm1_3(3,13),ptsm1_3(3,14),ptsm1_3(3,15); ...
    ptsm1_3(3,9),ptsm1_3(3,8),ptsm1_3(3,7),ptsm1_3(3,6), ...
    ptsm1_3(3,12),ptsm1_3(3,13),ptsm1_3(3,19),ptsm1_3(3,18), ...
    ptsm1_3(3,19),ptsm1_3(3,18),ptsm1_3(3,17),ptsm1_3(3,16); ...
    ptsm1_3(3,10),ptsm1_3(3,9),ptsm1_3(3,8),ptsm1_3(3,7), ...
    ptsm1_3(3,11),ptsm1_3(3,12),ptsm1_3(3,20),ptsm1_3(3,19), ...
    ptsm1_3(3,20),ptsm1_3(3,19),ptsm1_3(3,18),ptsm1_3(3,17)]];

 % Formation des matrices de position des points des formes définissant
l'effecteur 2

 xr=[xr,[ptse1(1,1),ptse1(1,2),ptse1(1,3),ptse1(1,1), ...
         ptse1(1,2),ptse1(1,3),ptse1(1,4),ptse1(1,8), ...
         ptse1(1,7),ptse1(1,6),ptse1(1,9),ptse1(1,10), ...
         ptse1(1,11); ...
         ptse1(1,2),ptse1(1,3),ptse1(1,4),ptse1(1,2), ...
         ptse1(1,3),ptse1(1,4),ptse1(1,5),ptse1(1,7), ...
         ptse1(1,6),ptse1(1,5),ptse1(1,10),ptse1(1,11), ...
         ptse1(1,12); ...
         ptse1(1,7),ptse1(1,6),ptse1(1,5),ptse1(1,10), ...
```

```
        ptse1(1,11),ptse1(1,12),ptse1(1,13),ptse1(1,15), ...
        ptse1(1,14),ptse1(1,13),ptse1(1,15),ptse1(1,14), ...
        ptse1(1,13); ...
        ptse1(1,8),ptse1(1,7),ptse1(1,6),ptse1(1,9), ...
        ptse1(1,10),ptse1(1,11),ptse1(1,12),ptse1(1,16), ...
        ptse1(1,15),ptse1(1,14),ptse1(1,16),ptse1(1,15), ...
        ptse1(1,14)]];

yr=[yr,[ptse1(2,1),ptse1(2,2),ptse1(2,3),ptse1(2,1), ...
        ptse1(2,2),ptse1(2,3),ptse1(2,4),ptse1(2,8), ...
        ptse1(2,7),ptse1(2,6),ptse1(2,9),ptse1(2,10), ...
        ptse1(2,11); ...
        ptse1(2,2),ptse1(2,3),ptse1(2,4),ptse1(2,2), ...
        ptse1(2,3),ptse1(2,4),ptse1(2,5),ptse1(2,7), ...
        ptse1(2,6),ptse1(2,5),ptse1(2,10),ptse1(2,11), ...
        ptse1(2,12); ...
        ptse1(2,7),ptse1(2,6),ptse1(2,5),ptse1(2,10), ...
        ptse1(2,11),ptse1(2,12),ptse1(2,13),ptse1(2,15), ...
        ptse1(2,14),ptse1(2,13),ptse1(2,15),ptse1(2,14), ...
        ptse1(2,13); ...
        ptse1(2,8),ptse1(2,7),ptse1(2,6),ptse1(2,9), ...
        ptse1(2,10),ptse1(2,11),ptse1(2,12),ptse1(2,16), ...
        ptse1(2,15),ptse1(2,14),ptse1(2,16),ptse1(2,15), ...
        ptse1(2,14)]];

zr=[zr,[ptse1(3,1),ptse1(3,2),ptse1(3,3),ptse1(3,1), ...
        ptse1(3,2),ptse1(3,3),ptse1(3,4),ptse1(3,8), ...
        ptse1(3,7),ptse1(3,6),ptse1(3,9),ptse1(3,10), ...
        ptse1(3,11); ...
        ptse1(3,2),ptse1(3,3),ptse1(3,4),ptse1(3,2), ...
        ptse1(3,3),ptse1(3,4),ptse1(3,5),ptse1(3,7), ...
        ptse1(3,6),ptse1(3,5),ptse1(3,10),ptse1(3,11), ...
        ptse1(3,12); ...
        ptse1(3,7),ptse1(3,6),ptse1(3,5),ptse1(3,10), ...
        ptse1(3,11),ptse1(3,12),ptse1(3,13),ptse1(3,15), ...
        ptse1(3,14),ptse1(3,13),ptse1(3,15),ptse1(3,14), ...
        ptse1(3,13); ...
        ptse1(3,8),ptse1(3,7),ptse1(3,6),ptse1(3,9), ...
        ptse1(3,10),ptse1(3,11),ptse1(3,12),ptse1(3,16), ...
        ptse1(3,15),ptse1(3,14),ptse1(3,16),ptse1(3,15), ...
        ptse1(3,14)]];

 % Formation des matrices de position des points des formes définissant
la base 3
 xr=[xr,[ptsb3(1,1),ptsb3(1,6),ptsb3(1,11),ptsb3(1,12),ptsb3(1,7); ...
     ptsb3(1,2),ptsb3(1,3),ptsb3(1,10),ptsb3(1,9),ptsb3(1,8); ...
     ptsb3(1,3),ptsb3(1,4),ptsb3(1,4),ptsb3(1,10),ptsb3(1,9); ...
     ptsb3(1,6),ptsb3(1,5),ptsb3(1,5),ptsb3(1,11),ptsb3(1,12)]];

 yr=[yr,[ptsb3(2,1),ptsb3(2,6),ptsb3(2,11),ptsb3(2,12),ptsb3(2,7); ...
     ptsb3(2,2),ptsb3(2,3),ptsb3(2,10),ptsb3(2,9),ptsb3(2,8); ...
     ptsb3(2,3),ptsb3(2,4),ptsb3(2,4),ptsb3(2,10),ptsb3(2,9); ...
     ptsb3(2,6),ptsb3(2,5),ptsb3(2,5),ptsb3(2,11),ptsb3(2,12)]];

 zr=[zr,[ptsb3(3,1),ptsb3(3,6),ptsb3(3,11),ptsb3(3,12),ptsb3(3,7); ...
     ptsb3(3,2),ptsb3(3,3),ptsb3(3,10),ptsb3(3,9),ptsb3(3,8); ...
     ptsb3(3,3),ptsb3(3,4),ptsb3(3,4),ptsb3(3,10),ptsb3(3,9); ...
```

```
    ptsb3(3,6),ptsb3(3,5),ptsb3(3,5),ptsb3(3,11),ptsb3(3,12)]];

 % Formation des matrices de position des points des formes définissant
le membre 1(doigt3)

 xr=[xr,[ptsm1_1(1,1),ptsm1_1(1,2),ptsm1_1(1,3),ptsm1_1(1,4), ...
    ptsm1_1(1,1),ptsm1_1(1,2),ptsm1_1(1,10),ptsm1_1(1,9), ...
    ptsm1_1(1,11),ptsm1_1(1,12),ptsm1_1(1,13),ptsm1_1(1,14); ...
    ptsm1_1(1,2),ptsm1_1(1,3),ptsm1_1(1,4),ptsm1_1(1,5), ...
    ptsm1_1(1,2),ptsm1_1(1,3),ptsm1_1(1,9),ptsm1_1(1,8), ...
    ptsm1_1(1,12),ptsm1_1(1,13),ptsm1_1(1,14),ptsm1_1(1,15); ...
    ptsm1_1(1,9),ptsm1_1(1,8),ptsm1_1(1,7),ptsm1_1(1,6), ...
    ptsm1_1(1,12),ptsm1_1(1,13),ptsm1_1(1,19),ptsm1_1(1,18), ...
    ptsm1_1(1,19),ptsm1_1(1,18),ptsm1_1(1,17),ptsm1_1(1,16); ...
    ptsm1_1(1,10),ptsm1_1(1,9),ptsm1_1(1,8),ptsm1_1(1,7), ...
    ptsm1_1(1,11),ptsm1_1(1,12),ptsm1_1(1,20),ptsm1_1(1,19), ...
    ptsm1_1(1,20),ptsm1_1(1,19),ptsm1_1(1,18),ptsm1_1(1,17)]];

 yr=[yr,[ptsm1_1(2,1),ptsm1_1(2,2),ptsm1_1(2,3),ptsm1_1(2,4), ...
    ptsm1_1(2,1),ptsm1_1(2,2),ptsm1_1(2,10),ptsm1_1(2,9), ...
    ptsm1_1(2,11),ptsm1_1(2,12),ptsm1_1(2,13),ptsm1_1(2,14); ...
    ptsm1_1(2,2),ptsm1_1(2,3),ptsm1_1(2,4),ptsm1_1(2,5), ...
    ptsm1_1(2,2),ptsm1_1(2,3),ptsm1_1(2,9),ptsm1_1(2,8), ...
    ptsm1_1(2,12),ptsm1_1(2,13),ptsm1_1(2,14),ptsm1_1(2,15); ...
    ptsm1_1(2,9),ptsm1_1(2,8),ptsm1_1(2,7),ptsm1_1(2,6), ...
    ptsm1_1(2,12),ptsm1_1(2,13),ptsm1_1(2,19),ptsm1_1(2,18), ...
    ptsm1_1(2,19),ptsm1_1(2,18),ptsm1_1(2,17),ptsm1_1(2,16); ...
    ptsm1_1(2,10),ptsm1_1(2,9),ptsm1_1(2,8),ptsm1_1(2,7), ...
    ptsm1_1(2,11),ptsm1_1(2,12),ptsm1_1(2,20),ptsm1_1(2,19), ...
    ptsm1_1(2,20),ptsm1_1(2,19),ptsm1_1(2,18),ptsm1_1(2,17)]];

 zr=[zr,[ptsm1_1(3,1),ptsm1_1(3,2),ptsm1_1(3,3),ptsm1_1(3,4), ...
    ptsm1_1(3,1),ptsm1_1(3,2),ptsm1_1(3,10),ptsm1_1(3,9), ...
    ptsm1_1(3,11),ptsm1_1(3,12),ptsm1_1(3,13),ptsm1_1(3,14); ...
    ptsm1_1(3,2),ptsm1_1(3,3),ptsm1_1(3,4),ptsm1_1(3,5), ...
    ptsm1_1(3,2),ptsm1_1(3,3),ptsm1_1(3,9),ptsm1_1(3,8), ...
    ptsm1_1(3,12),ptsm1_1(3,13),ptsm1_1(3,14),ptsm1_1(3,15); ...
    ptsm1_1(3,9),ptsm1_1(3,8),ptsm1_1(3,7),ptsm1_1(3,6), ...
    ptsm1_1(3,12),ptsm1_1(3,13),ptsm1_1(3,19),ptsm1_1(3,18), ...
    ptsm1_1(3,19),ptsm1_1(3,18),ptsm1_1(3,17),ptsm1_1(3,16); ...
    ptsm1_1(3,10),ptsm1_1(3,9),ptsm1_1(3,8),ptsm1_1(3,7), ...
    ptsm1_1(3,11),ptsm1_1(3,12),ptsm1_1(3,20),ptsm1_1(3,19), ...
    ptsm1_1(3,20),ptsm1_1(3,19),ptsm1_1(3,18),ptsm1_1(3,17)]];

 % Formation des matrices de position des points des formes définissant
le membre 2(doigt3)

 xr=[xr,[ptsm1_2(1,1),ptsm1_2(1,2),ptsm1_2(1,3),ptsm1_2(1,4), ...
    ptsm1_2(1,1),ptsm1_2(1,2),ptsm1_2(1,10),ptsm1_2(1,9), ...
    ptsm1_2(1,11),ptsm1_2(1,12),ptsm1_2(1,13),ptsm1_2(1,14); ...
    ptsm1_2(1,2),ptsm1_2(1,3),ptsm1_2(1,4),ptsm1_2(1,5), ...
    ptsm1_2(1,2),ptsm1_2(1,3),ptsm1_2(1,9),ptsm1_2(1,8), ...
    ptsm1_2(1,12),ptsm1_2(1,13),ptsm1_2(1,14),ptsm1_2(1,15); ...
    ptsm1_2(1,9),ptsm1_2(1,8),ptsm1_2(1,7),ptsm1_2(1,6), ...
    ptsm1_2(1,12),ptsm1_2(1,13),ptsm1_2(1,19),ptsm1_2(1,18), ...
    ptsm1_2(1,19),ptsm1_2(1,18),ptsm1_2(1,17),ptsm1_2(1,16); ...
    ptsm1_2(1,10),ptsm1_2(1,9),ptsm1_2(1,8),ptsm1_2(1,7), ...
```

```
    ptsm1_2(1,11),ptsm1_2(1,12),ptsm1_2(1,20),ptsm1_2(1,19), ...
    ptsm1_2(1,20),ptsm1_2(1,19),ptsm1_2(1,18),ptsm1_2(1,17)]];

 yr=[yr,[ptsm1_2(2,1),ptsm1_2(2,2),ptsm1_2(2,3),ptsm1_2(2,4), ...
    ptsm1_2(2,1),ptsm1_2(2,2),ptsm1_2(2,10),ptsm1_2(2,9), ...
    ptsm1_2(2,11),ptsm1_2(2,12),ptsm1_2(2,13),ptsm1_2(2,14); ...
    ptsm1_2(2,2),ptsm1_2(2,3),ptsm1_2(2,4),ptsm1_2(2,5), ...
    ptsm1_2(2,2),ptsm1_2(2,3),ptsm1_2(2,9),ptsm1_2(2,8), ...
    ptsm1_2(2,12),ptsm1_2(2,13),ptsm1_2(2,14),ptsm1_2(2,15); ...
    ptsm1_2(2,9),ptsm1_2(2,8),ptsm1_2(2,7),ptsm1_2(2,6), ...
    ptsm1_2(2,12),ptsm1_2(2,13),ptsm1_2(2,19),ptsm1_2(2,18), ...
    ptsm1_2(2,19),ptsm1_2(2,18),ptsm1_2(2,17),ptsm1_2(2,16); ...
    ptsm1_2(2,10),ptsm1_2(2,9),ptsm1_2(2,8),ptsm1_2(2,7), ...
    ptsm1_2(2,11),ptsm1_2(2,12),ptsm1_2(2,20),ptsm1_2(2,19), ...
    ptsm1_2(2,20),ptsm1_2(2,19),ptsm1_2(2,18),ptsm1_2(2,17)]];

 zr=[zr,[ptsm1_2(3,1),ptsm1_2(3,2),ptsm1_2(3,3),ptsm1_2(3,4), ...
    ptsm1_2(3,1),ptsm1_2(3,2),ptsm1_2(3,10),ptsm1_2(3,9), ...
    ptsm1_2(3,11),ptsm1_2(3,12),ptsm1_2(3,13),ptsm1_2(3,14); ...
    ptsm1_2(3,2),ptsm1_2(3,3),ptsm1_2(3,4),ptsm1_2(3,5), ...
    ptsm1_2(3,2),ptsm1_2(3,3),ptsm1_2(3,9),ptsm1_2(3,8), ...
    ptsm1_2(3,12),ptsm1_2(3,13),ptsm1_2(3,14),ptsm1_2(3,15); ...
    ptsm1_2(3,9),ptsm1_2(3,8),ptsm1_2(3,7),ptsm1_2(3,6), ...
    ptsm1_2(3,12),ptsm1_2(3,13),ptsm1_2(3,19),ptsm1_2(3,18), ...
    ptsm1_2(3,19),ptsm1_2(3,18),ptsm1_2(3,17),ptsm1_2(3,16); ...
    ptsm1_2(3,10),ptsm1_2(3,9),ptsm1_2(3,8),ptsm1_2(3,7), ...
    ptsm1_2(3,11),ptsm1_2(3,12),ptsm1_2(3,20),ptsm1_2(3,19), ...
    ptsm1_2(3,20),ptsm1_2(3,19),ptsm1_2(3,18),ptsm1_2(3,17)]];

 % Formation des matrices de position des points des formes définissant
le membre 3(doigt3)

 xr=[xr,[ptsm1_3(1,1),ptsm1_3(1,2),ptsm1_3(1,3),ptsm1_3(1,4), ...
    ptsm1_3(1,1),ptsm1_3(1,2),ptsm1_3(1,10),ptsm1_3(1,9), ...
    ptsm1_3(1,11),ptsm1_3(1,12),ptsm1_3(1,13),ptsm1_3(1,14); ...
    ptsm1_3(1,2),ptsm1_3(1,3),ptsm1_3(1,4),ptsm1_3(1,5), ...
    ptsm1_3(1,2),ptsm1_3(1,3),ptsm1_3(1,9),ptsm1_3(1,8), ...
    ptsm1_3(1,12),ptsm1_3(1,13),ptsm1_3(1,14),ptsm1_3(1,15); ...
    ptsm1_3(1,9),ptsm1_3(1,8),ptsm1_3(1,7),ptsm1_3(1,6), ...
    ptsm1_3(1,12),ptsm1_3(1,13),ptsm1_3(1,19),ptsm1_3(1,18), ...
    ptsm1_3(1,19),ptsm1_3(1,18),ptsm1_3(1,17),ptsm1_3(1,16); ...
    ptsm1_3(1,10),ptsm1_3(1,9),ptsm1_3(1,8),ptsm1_3(1,7), ...
    ptsm1_3(1,11),ptsm1_3(1,12),ptsm1_3(1,20),ptsm1_3(1,19), ...
    ptsm1_3(1,20),ptsm1_3(1,19),ptsm1_3(1,18),ptsm1_3(1,17)]];

 yr=[yr,[ptsm1_3(2,1),ptsm1_3(2,2),ptsm1_3(2,3),ptsm1_3(2,4), ...
    ptsm1_3(2,1),ptsm1_3(2,2),ptsm1_3(2,10),ptsm1_3(2,9), ...
    ptsm1_3(2,11),ptsm1_3(2,12),ptsm1_3(2,13),ptsm1_3(2,14); ...
    ptsm1_3(2,2),ptsm1_3(2,3),ptsm1_3(2,4),ptsm1_3(2,5), ...
    ptsm1_3(2,2),ptsm1_3(2,3),ptsm1_3(2,9),ptsm1_3(2,8), ...
    ptsm1_3(2,12),ptsm1_3(2,13),ptsm1_3(2,14),ptsm1_3(2,15); ...
    ptsm1_3(2,9),ptsm1_3(2,8),ptsm1_3(2,7),ptsm1_3(2,6), ...
    ptsm1_3(2,12),ptsm1_3(2,13),ptsm1_3(2,19),ptsm1_3(2,18), ...
    ptsm1_3(2,19),ptsm1_3(2,18),ptsm1_3(2,17),ptsm1_3(2,16); ...
    ptsm1_3(2,10),ptsm1_3(2,9),ptsm1_3(2,8),ptsm1_3(2,7), ...
    ptsm1_3(2,11),ptsm1_3(2,12),ptsm1_3(2,20),ptsm1_3(2,19), ...
    ptsm1_3(2,20),ptsm1_3(2,19),ptsm1_3(2,18),ptsm1_3(2,17)]];
```

```
zr=[zr,[ptsm1_3(3,1),ptsm1_3(3,2),ptsm1_3(3,3),ptsm1_3(3,4), ...
   ptsm1_3(3,1),ptsm1_3(3,2),ptsm1_3(3,10),ptsm1_3(3,9), ...
   ptsm1_3(3,11),ptsm1_3(3,12),ptsm1_3(3,13),ptsm1_3(3,14); ...
   ptsm1_3(3,2),ptsm1_3(3,3),ptsm1_3(3,4),ptsm1_3(3,5), ...
   ptsm1_3(3,2),ptsm1_3(3,3),ptsm1_3(3,9),ptsm1_3(3,8), ...
   ptsm1_3(3,12),ptsm1_3(3,13),ptsm1_3(3,14),ptsm1_3(3,15); ...
   ptsm1_3(3,9),ptsm1_3(3,8),ptsm1_3(3,7),ptsm1_3(3,6), ...
   ptsm1_3(3,12),ptsm1_3(3,13),ptsm1_3(3,19),ptsm1_3(3,18), ...
   ptsm1_3(3,19),ptsm1_3(3,18),ptsm1_3(3,17),ptsm1_3(3,16); ...
   ptsm1_3(3,10),ptsm1_3(3,9),ptsm1_3(3,8),ptsm1_3(3,7), ...
   ptsm1_3(3,11),ptsm1_3(3,12),ptsm1_3(3,20),ptsm1_3(3,19), ...
   ptsm1_3(3,20),ptsm1_3(3,19),ptsm1_3(3,18),ptsm1_3(3,17)]];

 % Formation des matrices de position des points des formes définissant
l'effecteur 3

 xr=[xr,[ptse1(1,1),ptse1(1,2),ptse1(1,3),ptse1(1,1), ...
         ptse1(1,2),ptse1(1,3),ptse1(1,4),ptse1(1,8), ...
         ptse1(1,7),ptse1(1,6),ptse1(1,9),ptse1(1,10), ...
         ptse1(1,11); ...
         ptse1(1,2),ptse1(1,3),ptse1(1,4),ptse1(1,2), ...
         ptse1(1,3),ptse1(1,4),ptse1(1,5),ptse1(1,7), ...
         ptse1(1,6),ptse1(1,5),ptse1(1,10),ptse1(1,11), ...
         ptse1(1,12); ...
         ptse1(1,7),ptse1(1,6),ptse1(1,5),ptse1(1,10), ...
         ptse1(1,11),ptse1(1,12),ptse1(1,13),ptse1(1,15), ...
         ptse1(1,14),ptse1(1,13),ptse1(1,15),ptse1(1,14), ...
         ptse1(1,13); ...
         ptse1(1,8),ptse1(1,7),ptse1(1,6),ptse1(1,9), ...
         ptse1(1,10),ptse1(1,11),ptse1(1,12),ptse1(1,16), ...
         ptse1(1,15),ptse1(1,14),ptse1(1,16),ptse1(1,15), ...
         ptse1(1,14)]];

 yr=[yr,[ptse1(2,1),ptse1(2,2),ptse1(2,3),ptse1(2,1), ...
         ptse1(2,2),ptse1(2,3),ptse1(2,4),ptse1(2,8), ...
         ptse1(2,7),ptse1(2,6),ptse1(2,9),ptse1(2,10), ...
         ptse1(2,11); ...
         ptse1(2,2),ptse1(2,3),ptse1(2,4),ptse1(2,2), ...
         ptse1(2,3),ptse1(2,4),ptse1(2,5),ptse1(2,7), ...
         ptse1(2,6),ptse1(2,5),ptse1(2,10),ptse1(2,11), ...
         ptse1(2,12); ...
         ptse1(2,7),ptse1(2,6),ptse1(2,5),ptse1(2,10), ...
         ptse1(2,11),ptse1(2,12),ptse1(2,13),ptse1(2,15), ...
         ptse1(2,14),ptse1(2,13),ptse1(2,15),ptse1(2,14), ...
         ptse1(2,13); ...
         ptse1(2,8),ptse1(2,7),ptse1(2,6),ptse1(2,9), ...
         ptse1(2,10),ptse1(2,11),ptse1(2,12),ptse1(2,16), ...
         ptse1(2,15),ptse1(2,14),ptse1(2,16),ptse1(2,15), ...
         ptse1(2,14)]];

 zr=[zr,[ptse1(3,1),ptse1(3,2),ptse1(3,3),ptse1(3,1), ...
         ptse1(3,2),ptse1(3,3),ptse1(3,4),ptse1(3,8), ...
         ptse1(3,7),ptse1(3,6),ptse1(3,9),ptse1(3,10), ...
         ptse1(3,11); ...
         ptse1(3,2),ptse1(3,3),ptse1(3,4),ptse1(3,2), ...
         ptse1(3,3),ptse1(3,4),ptse1(3,5),ptse1(3,7), ...
```

```
        ptsel(3,6),ptsel(3,5),ptsel(3,10),ptsel(3,11), ...
        ptsel(3,12); ...
        ptsel(3,7),ptsel(3,6),ptsel(3,5),ptsel(3,10), ...
        ptsel(3,11),ptsel(3,12),ptsel(3,13),ptsel(3,15), ...
        ptsel(3,14),ptsel(3,13),ptsel(3,15),ptsel(3,14), ...
        ptsel(3,13); ...
        ptsel(3,8),ptsel(3,7),ptsel(3,6),ptsel(3,9), ...
        ptsel(3,10),ptsel(3,11),ptsel(3,12),ptsel(3,16), ...
        ptsel(3,15),ptsel(3,14),ptsel(3,16),ptsel(3,15), ...
        ptsel(3,14)]];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%  Fin du calcul des points du tracé 3d   %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Début du tracé 3d %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 h1=fill3(xr,yr,zr,[cb cm1 cm2 cm3 ce cb cm1 cm2 cm3 ce cb cm1 cm2 cm3
ce]);
 hold on
 set(ha(6),'Visible','off');


 xlim([-1.7 1.7]);
 ylim([-1.7 1.7]);
 zlim([-1.7 1.7]);

 xyz=zeros(3,648);
 for a=1:162
    xyz(1:3,(a-1)*4+1:4*a)=[xr(:,a)';yr(:,a)';zr(:,a)'];
 end

 r3s2=sqrt(3)/2; %radical(3)/2

 %
 % store the block's handle in the figure's UserData
 %
 ud.Block=gcbh;
 ud.h1=h1;
 ud.xyz=xyz;
 ud.r3s2=r3s2;

 %
 % Associate the figure with the block, and set the figure's UserData.
 %
 SetSfunXYFigure(gcbh,FigHandle);
 set(FigHandle,'UserData',ud);


else
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%

h1=ud.h1;
```

```
xyz=ud.xyz;
r3s2=ud.r3s2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% Rafraichissement de la figure  %%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   c1=cos(u(1));
   s1=sin(u(1));
   c12=cos(u(1)+u(2));
   s12=sin(u(1)+u(2));
   c123=cos(u(1)+u(2)+u(3));
   s123=sin(u(1)+u(2)+u(3));
   Riv01=[c1,-s1,0;0,0,-1;s1,c1,0; ...
         c12,-s12,0;0,0,-1;s12,c12,0; ...
         c123,-s123,0;0,0,-1;s123,c123,0; ...
         -s123,-c123,0;0,0,-1;c123,-s123,0];
   Pid01=[u(10);0;0; ...
         u(10)+c1*u(11);0;s1*u(11); ...
         c12*u(12)+c1*u(11)+u(10);0;s12*u(12)+s1*u(11); ...

c123*u(13)+c12*u(12)+c1*u(11)+u(10);0;s123*u(13)+s12*u(12)+s1*u(11)];

   c1=cos(u(4));
   s1=sin(u(4));
   c12=cos(u(4)+u(5));
   s12=sin(u(4)+u(5));
   c123=cos(u(4)+u(5)+u(6));
   s123=sin(u(4)+u(5)+u(6));
   Riv02=[-0.5*c1,0.5*s1,r3s2;r3s2*c1,-r3s2*s1,0.5;s1,c1,0; ...
         -0.5*c12,0.5*s12,r3s2;r3s2*c12,-r3s2*s12,0.5;s12,c12,0; ...
         -0.5*c123,0.5*s123,r3s2;r3s2*c123,-r3s2*s123,0.5;s123,c123,0;
...
         0.5*s123,0.5*c123,r3s2;-r3s2*s123,-r3s2*c123,0.5;c123,-s123,0];
   Pid02=[-0.5*u(10);r3s2*u(10);0; ...
         -0.5*u(10)-0.5*c1*u(11);r3s2*(c1*u(11)+u(10));s1*u(11); ...
         -0.5*c12*u(12)-0.5*c1*u(11)-
0.5*u(10);r3s2*(c12*u(12)+c1*u(11)+u(10));s12*u(12)+s1*u(11); ...
         -0.5*c123*u(13)-0.5*c12*u(12)-0.5*c1*u(11)-
0.5*u(10);r3s2*(c123*u(13)+c12*u(12)+c1*u(11)+u(10));s123*u(13)+s12*u(12
)+s1*u(11)];

   c1=cos(u(7));
   s1=sin(u(7));
   c12=cos(u(7)+u(8));
   s12=sin(u(7)+u(8));
   c123=cos(u(7)+u(8)+u(9));
   s123=sin(u(7)+u(8)+u(9));
   Riv03=[-0.5*c1,0.5*s1,-r3s2;-r3s2*c1,r3s2*s1,0.5;s1,c1,0; ...
         -0.5*c12,0.5*s12,-r3s2;-r3s2*c12,r3s2*s12,0.5;s12,c12,0; ...
         -0.5*c123,0.5*s123,-r3s2;-r3s2*c123,r3s2*s123,0.5;s123,c123,0;
...
         0.5*s123,0.5*c123,-r3s2;r3s2*s123,r3s2*c123,0.5;c123,-s123,0];
   Pid03=[-0.5*u(10);-r3s2*u(10);0; ...
         -0.5*u(10)-0.5*c1*u(11);-r3s2*(c1*u(11)+u(10));s1*u(11); ...
         -0.5*c12*u(12)-0.5*c1*u(11)-0.5*u(10);-
r3s2*(c12*u(12)+c1*u(11)+u(10));s12*u(12)+s1*u(11); ...
```

```
        -0.5*c123*u(13)-0.5*c12*u(12)-0.5*c1*u(11)-0.5*u(10);-
r3s2*(c123*u(13)+c12*u(12)+c1*u(11)+u(10));s123*u(13)+s12*u(12)+s1*u(11)
];

   %Calcul des nouveaux points pour affichage(21:216)
   xyzr=[ xyz(:,1:20) ...
          Riv01(1:3,:)*xyz(:,21:68)+Pid01(1:3,1)*ones(1,48) ...
          Riv01(4:6,:)*xyz(:,69:116)+Pid01(4:6,1)*ones(1,48) ...
          Riv01(7:9,:)*xyz(:,117:164)+Pid01(7:9,1)*ones(1,48) ...
          Riv01(10:12,:)*xyz(:,165:216)+Pid01(10:12,1)*ones(1,52) ...
          xyz(:,217:236) ...
          Riv02(1:3,:)*xyz(:,237:284)+Pid02(1:3,1)*ones(1,48) ...
          Riv02(4:6,:)*xyz(:,285:332)+Pid02(4:6,1)*ones(1,48) ...
          Riv02(7:9,:)*xyz(:,333:380)+Pid02(7:9,1)*ones(1,48) ...
          Riv02(10:12,:)*xyz(:,381:432)+Pid02(10:12,1)*ones(1,52) ...
          xyz(:,433:452) ...
          Riv03(1:3,:)*xyz(:,453:500)+Pid03(1:3,1)*ones(1,48) ...
          Riv03(4:6,:)*xyz(:,501:548)+Pid03(4:6,1)*ones(1,48) ...
          Riv03(7:9,:)*xyz(:,549:596)+Pid03(7:9,1)*ones(1,48) ...
          Riv03(10:12,:)*xyz(:,597:648)+Pid03(10:12,1)*ones(1,52)];

   %Réorganisation des points pour afficher (216)

xr2=[xyzr(1,1:4:648);xyzr(1,2:4:648);xyzr(1,3:4:648);xyzr(1,4:4:648)];

yr2=[xyzr(2,1:4:648);xyzr(2,2:4:648);xyzr(2,3:4:648);xyzr(2,4:4:648)];

zr2=[xyzr(3,1:4:648);xyzr(3,2:4:648);xyzr(3,3:4:648);xyzr(3,4:4:648)];


   set(h1,{'xdata'},num2cell(xr2(:,1:162),1)', ...
      {'ydata'},num2cell(yr2(:,1:162),1)', ...
      {'zdata'},num2cell(zr2(:,1:162),1)');
   drawnow
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%% si handle...
% end mdlUpdate

%
%=============================================================================
======
% GetSfunXYFigure
% Retrieves the figure window associated with this S-function XY Graph
block
% from the block's parent subsystem's UserData.
%=============================================================================
======
%
function FigHandle=GetSfunXYFigure(block)

if strcmp(get_param(block,'BlockType'),'S-Function'),
  block=get_param(block,'Parent');
end

FigHandle=get_param(block,'UserData');
if isempty(FigHandle),
```

```
  FigHandle=-1;
end

% end GetSfunXYFigure

%
%=====================================================================
======
% SetSfunXYFigure
% Stores the figure window associated with this S-function XY Graph
block
% in the block's parent subsystem's UserData.
%=====================================================================
======
%
function SetSfunXYFigure(block,FigHandle)

if strcmp(get_param(bdroot,'BlockDiagramType'),'model'),
  if strcmp(get_param(block,'BlockType'),'S-Function'),
    block=get_param(block,'Parent');
  end

  set_param(block,'UserData',FigHandle);
end
```

Fonction (fichier script de Matlab) d'animation en 3 dimensions du manipulateur en temps différé

```
%animdif3d.m
%
% Programme d'animation 3D en temps différé
%
% Par : Martin de Montigny
%       Pierre Sicard
%
% Aout 1998
% Opal-rt
% UQTR
%
% Avant d'exécuter cette animation, il faut charger en mémoire les
résultats de simulation
% du doigt (modèle Lagrange-Euler, RRR)
%
% Certains paramètres de l'animation peuvent être ajustés comme :
% renderer : méthode que Matlab emploie pour dessiner (Open-gl, z-
buffers ou painters)
% projection : Matlab tient compte de la perspective ou non
% drawmode : accélération de l'animation au prix de la perte de certains
détails

% Chargement des résultats de la simulation
```

```
% load res280798a

% Reconstitution des matrices de rotation et des vecteurs de positions
des membres
Riv0=zeros(3*N,3);
Pid0=zeros(3*N,1);
for a=1:N
   Riv0((a-1)*3+1:(a-1)*3+3,:)=[rp(1,(a-1)*12+1:(a-1)*12+3); ...
      rp(1,(a-1)*12+4:(a-1)*12+6);rp(1,(a-1)*12+7:(a-1)*12+9)];
   Pid0((a-1)*3+1:(a-1)*3+3)=rp(1,(a-1)*12+10:(a-1)*12+12);
end

% test d'animation 3d du doigt à partir de données déja calculées (rp et
qi)

h=1;%!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!
visu2;
elevt=30;
azimt=30;
set(h,'renderer','painters')
ha=get(h,'children');
set(ha(3),'projection','perspective');%mettre un
tag.....................................
set(ha(3),'drawmode','fast');

% considérer les matrices de rotation des bases commes fixes...
rb1=eye(3);
pb1=[0;0;0];
for a=1:(length(qi)-1)
   rb1=[rb1;eye(3)];
   pb1=[pb1;[0;0;0]];
end

% Calcul des points de la base
larg=(L1+L2+L3+L4)/4*0.8;
ptsb1=[-larg/20,larg/20,larg/10;larg/20,larg/20,larg/10; ...
        larg/10,0,larg/10;larg/10,-larg/5,larg/10; ...
        -larg/10,-larg/5,larg/10;-larg/10,0,larg/10; ...
        -larg/20,larg/20,-larg/10;larg/20,larg/20,-larg/10; ...
        larg/10,0,-larg/10;larg/10,-larg/5,-larg/10; ...
        -larg/10,-larg/5,-larg/10;-larg/10,0,-larg/10; ...
        0,0,larg/10+larg/100;0,0,-larg/10-larg/100];

% Calcul des points des membres
ptsm1=[-larg/20,larg/20,-larg*0.07;0,larg/10,-larg*0.07; ...
       L1-larg/5,larg/10,-larg/10;L1,larg/10,-larg/10; ...
       L1+larg/20,larg/20,-larg/10;L1+larg/20,-larg/20,-larg/10; ...
       L1,-larg/10,-larg/10;L1-larg/5,-larg/10,-larg/10; ...
       0,-larg/10,-larg*0.07;-larg/20,-larg/20,-larg*0.07; ...
       -larg/20,larg/20,larg*0.07;0,larg/10,larg*0.07; ...
       L1-larg/5,larg/10,larg/10;L1,larg/10,larg/10; ...
       L1+larg/20,larg/20,larg/10;L1+larg/20,-larg/20,larg/10; ...
       L1,-larg/10,larg/10;L1-larg/5,-larg/10,larg/10; ...
       0,-larg/10,larg*0.07;-larg/20,-larg/20,larg*0.07];
ptsm2=[-larg/20,larg/20,-larg*0.07;0,larg/10,-larg*0.07; ...
       L2-larg/5,larg/10,-larg/10;L2,larg/10,-larg/10; ...
```

```
       L2+larg/20,larg/20,-larg/10;L2+larg/20,-larg/20,-larg/10; ...
       L2,-larg/10,-larg/10;L2-larg/5,-larg/10,-larg/10; ...
       0,-larg/10,-larg*0.07;-larg/20,-larg/20,-larg*0.07; ...
       -larg/20,larg/20,larg*0.07;0,larg/10,larg*0.07; ...
       L2-larg/5,larg/10,larg/10;L2,larg/10,larg/10; ...
       L2+larg/20,larg/20,larg/10;L2+larg/20,-larg/20,larg/10; ...
       L2,-larg/10,larg/10;L2-larg/5,-larg/10,larg/10; ...
       0,-larg/10,larg*0.07;-larg/20,-larg/20,larg*0.07];
ptsm3=[-larg/20,larg/20,-larg*0.07;0,larg/10,-larg*0.07; ...
       L3-larg/5,larg/10,-larg/10;L3,larg/10,-larg/10; ...
       L3+larg/20,larg/20,-larg/10;L3+larg/20,-larg/20,-larg/10; ...
       L3,-larg/10,-larg/10;L3-larg/5,-larg/10,-larg/10; ...
       0,-larg/10,-larg*0.07;-larg/20,-larg/20,-larg*0.07; ...
       -larg/20,larg/20,larg*0.07;0,larg/10,larg*0.07; ...
       L3-larg/5,larg/10,larg/10;L3,larg/10,larg/10; ...
       L3+larg/20,larg/20,larg/10;L3+larg/20,-larg/20,larg/10; ...
       L3,-larg/10,larg/10;L3-larg/5,-larg/10,larg/10; ...
       0,-larg/10,larg*0.07;-larg/20,-larg/20,larg*0.07];

%calcul des points de l'effecteur
ptse1=[-larg/20,larg/20,-larg*0.07;0,larg/10,-larg*0.07; ...
       L4-larg/20,larg/10,-larg*0.07;L4,larg/20,-larg/20; ...
       L4,-larg/20,-larg/20;L4-larg/20,-larg/10,-larg*0.07; ...
       0,-larg/10,-larg*0.07;-larg/20,-larg/20,-larg*0.07; ...
       -larg/20,larg/20,larg*0.07;0,larg/10,larg*0.07; ...
       L4-larg/20,larg/10,larg*0.07;L4,larg/20,larg/20; ...
       L4,-larg/20,larg/20;L4-larg/20,-larg/10,larg*0.07; ...
       0,-larg/10,larg*0.07;-larg/20,-larg/20,larg*0.07];

% Met le vecteur de points dans le bon sens pour la multiplication avec
les
% lignes de la matrice de rotation
ptsb1=ptsb1';
ptsm1_1=ptsm1';
%ptsm1_1x=Riv0(1,1:3)*ptsm1_1+Pid0(1); à ne pas faire car les rot.,
trans. se retrouvent en offset (en double)
%ptsm1_1y=Riv0(2,1:3)*ptsm1_1+Pid0(2); lors de la réassignation de la
position dans l'animation (les "set")
%ptsm1_1z=Riv0(3,1:3)*ptsm1_1+Pid0(3);
%ptsm1_1=[ptsm1_1x;ptsm1_1y;ptsm1_1z];
ptsm1_2=ptsm2';
%ptsm1_2x=Riv0(4,1:3)*ptsm1_2+Pid0(4);
%ptsm1_2y=Riv0(5,1:3)*ptsm1_2+Pid0(5);
%ptsm1_2z=Riv0(6,1:3)*ptsm1_2+Pid0(6);
%ptsm1_2=[ptsm1_2x;ptsm1_2y;ptsm1_2z];
ptsm1_3=ptsm3';
%ptsm1_3x=Riv0(7,1:3)*ptsm1_3+Pid0(7);
%ptsm1_3y=Riv0(8,1:3)*ptsm1_3+Pid0(8);
%ptsm1_3z=Riv0(9,1:3)*ptsm1_3+Pid0(9);
%ptsm1_3=[ptsm1_3x;ptsm1_3y;ptsm1_3z];
ptse1=ptse1';
%ptse1x=Riv0(10,1:3)*ptse1+Pid0(10);
%ptse1y=Riv0(11,1:3)*ptse1+Pid0(11);
%ptse1z=Riv0(12,1:3)*ptse1+Pid0(12);
%ptse1=[ptse1x;ptse1y;ptse1z];
```

```
% Formation des matrices de position des points des formes définissant
la base
xr=[ptsb1(1,1),ptsb1(1,6),ptsb1(1,11),ptsb1(1,12),ptsb1(1,7); ...
    ptsb1(1,2),ptsb1(1,3),ptsb1(1,10),ptsb1(1,9),ptsb1(1,8); ...
    ptsb1(1,3),ptsb1(1,4),ptsb1(1,4),ptsb1(1,10),ptsb1(1,9); ...
    ptsb1(1,6),ptsb1(1,5),ptsb1(1,5),ptsb1(1,11),ptsb1(1,12)];

yr=[ptsb1(2,1),ptsb1(2,6),ptsb1(2,11),ptsb1(2,12),ptsb1(2,7); ...
    ptsb1(2,2),ptsb1(2,3),ptsb1(2,10),ptsb1(2,9),ptsb1(2,8); ...
    ptsb1(2,3),ptsb1(2,4),ptsb1(2,4),ptsb1(2,10),ptsb1(2,9); ...
    ptsb1(2,6),ptsb1(2,5),ptsb1(2,5),ptsb1(2,11),ptsb1(2,12)];

zr=[ptsb1(3,1),ptsb1(3,6),ptsb1(3,11),ptsb1(3,12),ptsb1(3,7); ...
    ptsb1(3,2),ptsb1(3,3),ptsb1(3,10),ptsb1(3,9),ptsb1(3,8); ...
    ptsb1(3,3),ptsb1(3,4),ptsb1(3,4),ptsb1(3,10),ptsb1(3,9); ...
    ptsb1(3,6),ptsb1(3,5),ptsb1(3,5),ptsb1(3,11),ptsb1(3,12)];

cb=zeros(4,5); %couleur de la base

% Formation des matrices de position des points des formes définissant
le membre 1

xr=[xr,[ptsm1_1(1,1),ptsm1_1(1,2),ptsm1_1(1,3),ptsm1_1(1,4), ...
   ptsm1_1(1,1),ptsm1_1(1,2),ptsm1_1(1,10),ptsm1_1(1,9), ...
   ptsm1_1(1,11),ptsm1_1(1,12),ptsm1_1(1,13),ptsm1_1(1,14); ...
   ptsm1_1(1,2),ptsm1_1(1,3),ptsm1_1(1,4),ptsm1_1(1,5), ...
   ptsm1_1(1,2),ptsm1_1(1,3),ptsm1_1(1,9),ptsm1_1(1,8), ...
   ptsm1_1(1,12),ptsm1_1(1,13),ptsm1_1(1,14),ptsm1_1(1,15); ...
   ptsm1_1(1,9),ptsm1_1(1,8),ptsm1_1(1,7),ptsm1_1(1,6), ...
   ptsm1_1(1,12),ptsm1_1(1,13),ptsm1_1(1,19),ptsm1_1(1,18), ...
   ptsm1_1(1,19),ptsm1_1(1,18),ptsm1_1(1,17),ptsm1_1(1,16); ...
   ptsm1_1(1,10),ptsm1_1(1,9),ptsm1_1(1,8),ptsm1_1(1,7), ...
   ptsm1_1(1,11),ptsm1_1(1,12),ptsm1_1(1,20),ptsm1_1(1,19), ...
   ptsm1_1(1,20),ptsm1_1(1,19),ptsm1_1(1,18),ptsm1_1(1,17)]];

yr=[yr,[ptsm1_1(2,1),ptsm1_1(2,2),ptsm1_1(2,3),ptsm1_1(2,4), ...
   ptsm1_1(2,1),ptsm1_1(2,2),ptsm1_1(2,10),ptsm1_1(2,9), ...
   ptsm1_1(2,11),ptsm1_1(2,12),ptsm1_1(2,13),ptsm1_1(2,14); ...
   ptsm1_1(2,2),ptsm1_1(2,3),ptsm1_1(2,4),ptsm1_1(2,5), ...
   ptsm1_1(2,2),ptsm1_1(2,3),ptsm1_1(2,9),ptsm1_1(2,8), ...
   ptsm1_1(2,12),ptsm1_1(2,13),ptsm1_1(2,14),ptsm1_1(2,15); ...
   ptsm1_1(2,9),ptsm1_1(2,8),ptsm1_1(2,7),ptsm1_1(2,6), ...
   ptsm1_1(2,12),ptsm1_1(2,13),ptsm1_1(2,19),ptsm1_1(2,18), ...
   ptsm1_1(2,19),ptsm1_1(2,18),ptsm1_1(2,17),ptsm1_1(2,16); ...
   ptsm1_1(2,10),ptsm1_1(2,9),ptsm1_1(2,8),ptsm1_1(2,7), ...
   ptsm1_1(2,11),ptsm1_1(2,12),ptsm1_1(2,20),ptsm1_1(2,19), ...
   ptsm1_1(2,20),ptsm1_1(2,19),ptsm1_1(2,18),ptsm1_1(2,17)]];

zr=[zr,[ptsm1_1(3,1),ptsm1_1(3,2),ptsm1_1(3,3),ptsm1_1(3,4), ...
   ptsm1_1(3,1),ptsm1_1(3,2),ptsm1_1(3,10),ptsm1_1(3,9), ...
   ptsm1_1(3,11),ptsm1_1(3,12),ptsm1_1(3,13),ptsm1_1(3,14); ...
   ptsm1_1(3,2),ptsm1_1(3,3),ptsm1_1(3,4),ptsm1_1(3,5), ...
   ptsm1_1(3,2),ptsm1_1(3,3),ptsm1_1(3,9),ptsm1_1(3,8), ...
   ptsm1_1(3,12),ptsm1_1(3,13),ptsm1_1(3,14),ptsm1_1(3,15); ...
   ptsm1_1(3,9),ptsm1_1(3,8),ptsm1_1(3,7),ptsm1_1(3,6), ...
   ptsm1_1(3,12),ptsm1_1(3,13),ptsm1_1(3,19),ptsm1_1(3,18), ...
   ptsm1_1(3,19),ptsm1_1(3,18),ptsm1_1(3,17),ptsm1_1(3,16); ...
```

```
   ptsm1_1(3,10),ptsm1_1(3,9),ptsm1_1(3,8),ptsm1_1(3,7), ...
   ptsm1_1(3,11),ptsm1_1(3,12),ptsm1_1(3,20),ptsm1_1(3,19), ...
   ptsm1_1(3,20),ptsm1_1(3,19),ptsm1_1(3,18),ptsm1_1(3,17)]];

cm1=0.25*ones(4,12);

% Formation des matrices de position des points des formes définissant
le membre 2

xr=[xr,[ptsm1_2(1,1),ptsm1_2(1,2),ptsm1_2(1,3),ptsm1_2(1,4), ...
   ptsm1_2(1,1),ptsm1_2(1,2),ptsm1_2(1,10),ptsm1_2(1,9), ...
   ptsm1_2(1,11),ptsm1_2(1,12),ptsm1_2(1,13),ptsm1_2(1,14); ...
   ptsm1_2(1,2),ptsm1_2(1,3),ptsm1_2(1,4),ptsm1_2(1,5), ...
   ptsm1_2(1,2),ptsm1_2(1,3),ptsm1_2(1,9),ptsm1_2(1,8), ...
   ptsm1_2(1,12),ptsm1_2(1,13),ptsm1_2(1,14),ptsm1_2(1,15); ...
   ptsm1_2(1,9),ptsm1_2(1,8),ptsm1_2(1,7),ptsm1_2(1,6), ...
   ptsm1_2(1,12),ptsm1_2(1,13),ptsm1_2(1,19),ptsm1_2(1,18), ...
   ptsm1_2(1,19),ptsm1_2(1,18),ptsm1_2(1,17),ptsm1_2(1,16); ...
   ptsm1_2(1,10),ptsm1_2(1,9),ptsm1_2(1,8),ptsm1_2(1,7), ...
   ptsm1_2(1,11),ptsm1_2(1,12),ptsm1_2(1,20),ptsm1_2(1,19), ...
   ptsm1_2(1,20),ptsm1_2(1,19),ptsm1_2(1,18),ptsm1_2(1,17)]];

yr=[yr,[ptsm1_2(2,1),ptsm1_2(2,2),ptsm1_2(2,3),ptsm1_2(2,4), ...
   ptsm1_2(2,1),ptsm1_2(2,2),ptsm1_2(2,10),ptsm1_2(2,9), ...
   ptsm1_2(2,11),ptsm1_2(2,12),ptsm1_2(2,13),ptsm1_2(2,14); ...
   ptsm1_2(2,2),ptsm1_2(2,3),ptsm1_2(2,4),ptsm1_2(2,5), ...
   ptsm1_2(2,2),ptsm1_2(2,3),ptsm1_2(2,9),ptsm1_2(2,8), ...
   ptsm1_2(2,12),ptsm1_2(2,13),ptsm1_2(2,14),ptsm1_2(2,15); ...
   ptsm1_2(2,9),ptsm1_2(2,8),ptsm1_2(2,7),ptsm1_2(2,6), ...
   ptsm1_2(2,12),ptsm1_2(2,13),ptsm1_2(2,19),ptsm1_2(2,18), ...
   ptsm1_2(2,19),ptsm1_2(2,18),ptsm1_2(2,17),ptsm1_2(2,16); ...
   ptsm1_2(2,10),ptsm1_2(2,9),ptsm1_2(2,8),ptsm1_2(2,7), ...
   ptsm1_2(2,11),ptsm1_2(2,12),ptsm1_2(2,20),ptsm1_2(2,19), ...
   ptsm1_2(2,20),ptsm1_2(2,19),ptsm1_2(2,18),ptsm1_2(2,17)]];

zr=[zr,[ptsm1_2(3,1),ptsm1_2(3,2),ptsm1_2(3,3),ptsm1_2(3,4), ...
   ptsm1_2(3,1),ptsm1_2(3,2),ptsm1_2(3,10),ptsm1_2(3,9), ...
   ptsm1_2(3,11),ptsm1_2(3,12),ptsm1_2(3,13),ptsm1_2(3,14); ...
   ptsm1_2(3,2),ptsm1_2(3,3),ptsm1_2(3,4),ptsm1_2(3,5), ...
   ptsm1_2(3,2),ptsm1_2(3,3),ptsm1_2(3,9),ptsm1_2(3,8), ...
   ptsm1_2(3,12),ptsm1_2(3,13),ptsm1_2(3,14),ptsm1_2(3,15); ...
   ptsm1_2(3,9),ptsm1_2(3,8),ptsm1_2(3,7),ptsm1_2(3,6), ...
   ptsm1_2(3,12),ptsm1_2(3,13),ptsm1_2(3,19),ptsm1_2(3,18), ...
   ptsm1_2(3,19),ptsm1_2(3,18),ptsm1_2(3,17),ptsm1_2(3,16); ...
   ptsm1_2(3,10),ptsm1_2(3,9),ptsm1_2(3,8),ptsm1_2(3,7), ...
   ptsm1_2(3,11),ptsm1_2(3,12),ptsm1_2(3,20),ptsm1_2(3,19), ...
   ptsm1_2(3,20),ptsm1_2(3,19),ptsm1_2(3,18),ptsm1_2(3,17)]];

cm2=0.5*ones(4,12);


% Formation des matrices de position des points des formes définissant
le membre 3

xr=[xr,[ptsm1_3(1,1),ptsm1_3(1,2),ptsm1_3(1,3),ptsm1_3(1,4), ...
   ptsm1_3(1,1),ptsm1_3(1,2),ptsm1_3(1,10),ptsm1_3(1,9), ...
   ptsm1_3(1,11),ptsm1_3(1,12),ptsm1_3(1,13),ptsm1_3(1,14); ...
```

```
   ptsm1_3(1,2),ptsm1_3(1,3),ptsm1_3(1,4),ptsm1_3(1,5), ...
   ptsm1_3(1,2),ptsm1_3(1,3),ptsm1_3(1,9),ptsm1_3(1,8), ...
   ptsm1_3(1,12),ptsm1_3(1,13),ptsm1_3(1,14),ptsm1_3(1,15); ...
   ptsm1_3(1,9),ptsm1_3(1,8),ptsm1_3(1,7),ptsm1_3(1,6), ...
   ptsm1_3(1,12),ptsm1_3(1,13),ptsm1_3(1,19),ptsm1_3(1,18), ...
   ptsm1_3(1,19),ptsm1_3(1,18),ptsm1_3(1,17),ptsm1_3(1,16); ...
   ptsm1_3(1,10),ptsm1_3(1,9),ptsm1_3(1,8),ptsm1_3(1,7), ...
   ptsm1_3(1,11),ptsm1_3(1,12),ptsm1_3(1,20),ptsm1_3(1,19), ...
   ptsm1_3(1,20),ptsm1_3(1,19),ptsm1_3(1,18),ptsm1_3(1,17)]];

yr=[yr,[ptsm1_3(2,1),ptsm1_3(2,2),ptsm1_3(2,3),ptsm1_3(2,4), ...
   ptsm1_3(2,1),ptsm1_3(2,2),ptsm1_3(2,10),ptsm1_3(2,9), ...
   ptsm1_3(2,11),ptsm1_3(2,12),ptsm1_3(2,13),ptsm1_3(2,14); ...
   ptsm1_3(2,2),ptsm1_3(2,3),ptsm1_3(2,4),ptsm1_3(2,5), ...
   ptsm1_3(2,2),ptsm1_3(2,3),ptsm1_3(2,9),ptsm1_3(2,8), ...
   ptsm1_3(2,12),ptsm1_3(2,13),ptsm1_3(2,14),ptsm1_3(2,15); ...
   ptsm1_3(2,9),ptsm1_3(2,8),ptsm1_3(2,7),ptsm1_3(2,6), ...
   ptsm1_3(2,12),ptsm1_3(2,13),ptsm1_3(2,19),ptsm1_3(2,18), ...
   ptsm1_3(2,19),ptsm1_3(2,18),ptsm1_3(2,17),ptsm1_3(2,16); ...
   ptsm1_3(2,10),ptsm1_3(2,9),ptsm1_3(2,8),ptsm1_3(2,7), ...
   ptsm1_3(2,11),ptsm1_3(2,12),ptsm1_3(2,20),ptsm1_3(2,19), ...
   ptsm1_3(2,20),ptsm1_3(2,19),ptsm1_3(2,18),ptsm1_3(2,17)]];

zr=[zr,[ptsm1_3(3,1),ptsm1_3(3,2),ptsm1_3(3,3),ptsm1_3(3,4), ...
   ptsm1_3(3,1),ptsm1_3(3,2),ptsm1_3(3,10),ptsm1_3(3,9), ...
   ptsm1_3(3,11),ptsm1_3(3,12),ptsm1_3(3,13),ptsm1_3(3,14); ...
   ptsm1_3(3,2),ptsm1_3(3,3),ptsm1_3(3,4),ptsm1_3(3,5), ...
   ptsm1_3(3,2),ptsm1_3(3,3),ptsm1_3(3,9),ptsm1_3(3,8), ...
   ptsm1_3(3,12),ptsm1_3(3,13),ptsm1_3(3,14),ptsm1_3(3,15); ...
   ptsm1_3(3,9),ptsm1_3(3,8),ptsm1_3(3,7),ptsm1_3(3,6), ...
   ptsm1_3(3,12),ptsm1_3(3,13),ptsm1_3(3,19),ptsm1_3(3,18), ...
   ptsm1_3(3,19),ptsm1_3(3,18),ptsm1_3(3,17),ptsm1_3(3,16); ...
   ptsm1_3(3,10),ptsm1_3(3,9),ptsm1_3(3,8),ptsm1_3(3,7), ...
   ptsm1_3(3,11),ptsm1_3(3,12),ptsm1_3(3,20),ptsm1_3(3,19), ...
   ptsm1_3(3,20),ptsm1_3(3,19),ptsm1_3(3,18),ptsm1_3(3,17)]];

cm3=0.75*ones(4,12);


% Formation des matrices de position des points des formes définissant
l'effecteur

xr=[xr,[ptse1(1,1),ptse1(1,2),ptse1(1,3),ptse1(1,1), ...
        ptse1(1,2),ptse1(1,3),ptse1(1,4),ptse1(1,8), ...
        ptse1(1,7),ptse1(1,6),ptse1(1,9),ptse1(1,10), ...
        ptse1(1,11); ...
        ptse1(1,2),ptse1(1,3),ptse1(1,4),ptse1(1,2), ...
        ptse1(1,3),ptse1(1,4),ptse1(1,5),ptse1(1,7), ...
        ptse1(1,6),ptse1(1,5),ptse1(1,10),ptse1(1,11), ...
        ptse1(1,12); ...
        ptse1(1,7),ptse1(1,6),ptse1(1,5),ptse1(1,10), ...
        ptse1(1,11),ptse1(1,12),ptse1(1,13),ptse1(1,15), ...
        ptse1(1,14),ptse1(1,13),ptse1(1,15),ptse1(1,14), ...
        ptse1(1,13); ...
        ptse1(1,8),ptse1(1,7),ptse1(1,6),ptse1(1,9), ...
        ptse1(1,10),ptse1(1,11),ptse1(1,12),ptse1(1,16), ...
        ptse1(1,15),ptse1(1,14),ptse1(1,16),ptse1(1,15), ...
```

```
        ptse1(1,14)]];

yr=[yr,[ptse1(2,1),ptse1(2,2),ptse1(2,3),ptse1(2,1), ...
        ptse1(2,2),ptse1(2,3),ptse1(2,4),ptse1(2,8), ...
        ptse1(2,7),ptse1(2,6),ptse1(2,9),ptse1(2,10), ...
        ptse1(2,11); ...
        ptse1(2,2),ptse1(2,3),ptse1(2,4),ptse1(2,2), ...
        ptse1(2,3),ptse1(2,4),ptse1(2,5),ptse1(2,7), ...
        ptse1(2,6),ptse1(2,5),ptse1(2,10),ptse1(2,11), ...
        ptse1(2,12); ...
        ptse1(2,7),ptse1(2,6),ptse1(2,5),ptse1(2,10), ...
        ptse1(2,11),ptse1(2,12),ptse1(2,13),ptse1(2,15), ...
        ptse1(2,14),ptse1(2,13),ptse1(2,15),ptse1(2,14), ...
        ptse1(2,13); ...
        ptse1(2,8),ptse1(2,7),ptse1(2,6),ptse1(2,9), ...
        ptse1(2,10),ptse1(2,11),ptse1(2,12),ptse1(2,16), ...
        ptse1(2,15),ptse1(2,14),ptse1(2,16),ptse1(2,15), ...
        ptse1(2,14)]];

zr=[zr,[ptse1(3,1),ptse1(3,2),ptse1(3,3),ptse1(3,1), ...
        ptse1(3,2),ptse1(3,3),ptse1(3,4),ptse1(3,8), ...
        ptse1(3,7),ptse1(3,6),ptse1(3,9),ptse1(3,10), ...
        ptse1(3,11); ...
        ptse1(3,2),ptse1(3,3),ptse1(3,4),ptse1(3,2), ...
        ptse1(3,3),ptse1(3,4),ptse1(3,5),ptse1(3,7), ...
        ptse1(3,6),ptse1(3,5),ptse1(3,10),ptse1(3,11), ...
        ptse1(3,12); ...
        ptse1(3,7),ptse1(3,6),ptse1(3,5),ptse1(3,10), ...
        ptse1(3,11),ptse1(3,12),ptse1(3,13),ptse1(3,15), ...
        ptse1(3,14),ptse1(3,13),ptse1(3,15),ptse1(3,14), ...
        ptse1(3,13); ...
        ptse1(3,8),ptse1(3,7),ptse1(3,6),ptse1(3,9), ...
        ptse1(3,10),ptse1(3,11),ptse1(3,12),ptse1(3,16), ...
        ptse1(3,15),ptse1(3,14),ptse1(3,16),ptse1(3,15), ...
        ptse1(3,14)]];

ce=1*ones(4,13);


h1=fill3(xr,yr,zr,[cb cm1 cm2 cm3 ce]);
hold on

xlim([-0.5 0.5]);
ylim([-0.5 0.5]);
zlim([-0.5 0.5]);


tic;
for a=1:51 %nombre d'itérations
   for b=1:4 %nombre de points par polygone
      %p.s. : on ne redessine pas la base
      for c=6:17 %numéro des polygones à replacer
         Riv0(1:3,:)=[rp(a,1:3);rp(a,4:6);rp(a,7:9)];
         Pid0(1:3)=rp(a,10:12);
         xr2(b,c)=Riv0(1,:)*[xr(b,c);yr(b,c);zr(b,c)]+Pid0(1);
         yr2(b,c)=Riv0(2,:)*[xr(b,c);yr(b,c);zr(b,c)]+Pid0(2);
         zr2(b,c)=Riv0(3,:)*[xr(b,c);yr(b,c);zr(b,c)]+Pid0(3);
```

```
        end
        for c=18:29
           Riv0(4:6,:)=[rp(a,13:15);rp(a,16:18);rp(a,19:21)];
           Pid0(4:6)=rp(a,22:24);
           xr2(b,c)=Riv0(4,:)*[xr(b,c);yr(b,c);zr(b,c)]+Pid0(4);
           yr2(b,c)=Riv0(5,:)*[xr(b,c);yr(b,c);zr(b,c)]+Pid0(5);
           zr2(b,c)=Riv0(6,:)*[xr(b,c);yr(b,c);zr(b,c)]+Pid0(6);
        end
        for c=30:41
           Riv0(7:9,:)=[rp(a,25:27);rp(a,28:30);rp(a,31:33)];
           Pid0(7:9)=rp(a,34:36);
           xr2(b,c)=Riv0(7,:)*[xr(b,c);yr(b,c);zr(b,c)]+Pid0(7);
           yr2(b,c)=Riv0(8,:)*[xr(b,c);yr(b,c);zr(b,c)]+Pid0(8);
           zr2(b,c)=Riv0(9,:)*[xr(b,c);yr(b,c);zr(b,c)]+Pid0(9);
        end
        for c=42:54
           Riv0(10:12,:)=[rp(a,37:39);rp(a,40:42);rp(a,43:45)];
           Pid0(10:12)=rp(a,46:48);
           xr2(b,c)=Riv0(10,:)*[xr(b,c);yr(b,c);zr(b,c)]+Pid0(10);
           yr2(b,c)=Riv0(11,:)*[xr(b,c);yr(b,c);zr(b,c)]+Pid0(11);
           zr2(b,c)=Riv0(12,:)*[xr(b,c);yr(b,c);zr(b,c)]+Pid0(12);
        end
     end
     set(h1,{'xdata'},num2cell([xr(:,1:5),xr2(:,6:54)],1)', ...
        {'ydata'},num2cell([yr(:,1:5),yr2(:,6:54)],1)', ...
        {'zdata'},num2cell([zr(:,1:5),zr2(:,6:54)],1)');
     drawnow
  end
  toc
```

Annexe G

Résultats obtenus lors de l'étude de cas

Résultats obtenus pour les essais effectués dans l'étude de cas du chapitre V (les résultats sont identiques pour les trois doigts)

Résultats obtenus suite à l'essai 2 (Rigidité de l'objet de 500N/m et erreur cinématique de -5mm)

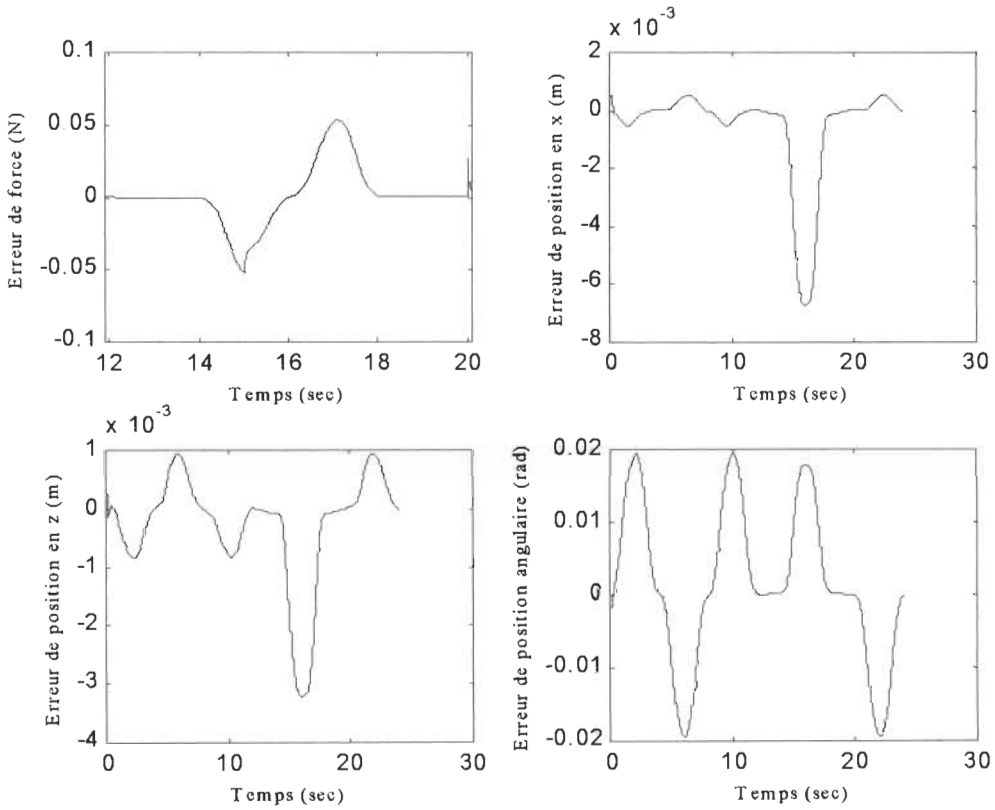

Figure G. 1 : Résultats obtenus suite à l'essai 2 (Rigidité de l'objet de 500N/m et erreur cinématique de -5mm)

Résultats obtenus suite à l'essai 4 (Rigidité de l'objet de 500N/m et erreur cinématique de 20mm)

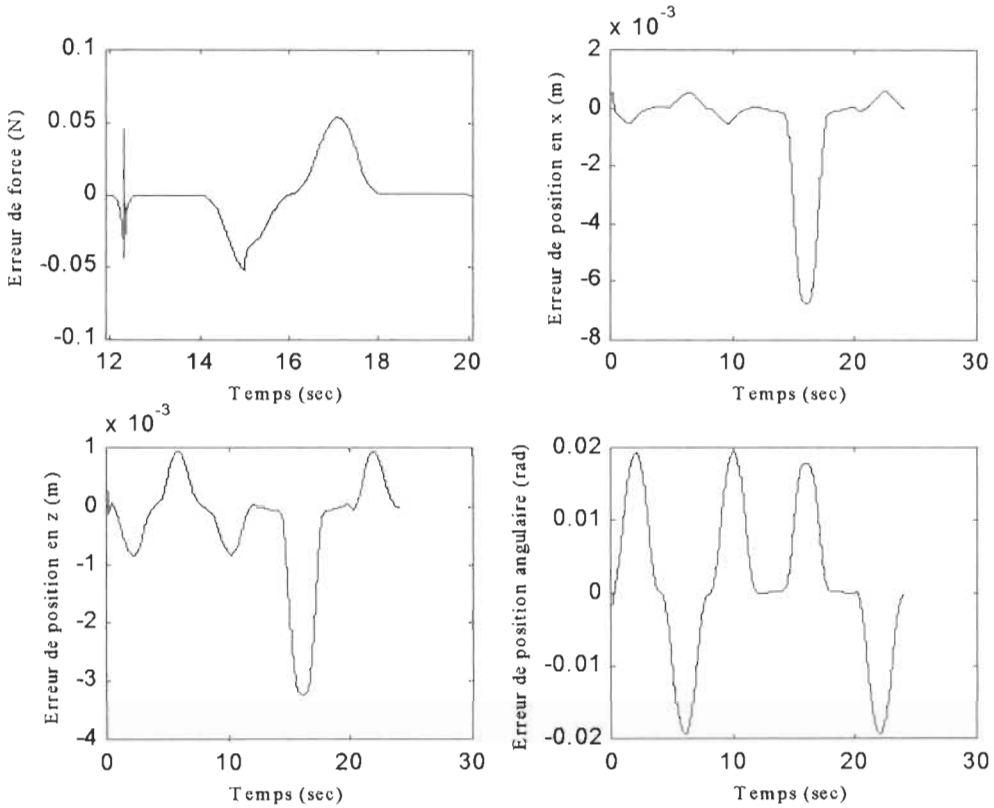

Figure G. 2 : Résultats obtenus suite à l'essai 4 (Rigidité de l'objet de 500N/m et erreur cinématique de 20mm)

Résultats obtenus suite à l'essai 6 (Rigidité de l'objet de 1kN/m et erreur cinématique de -5mm)

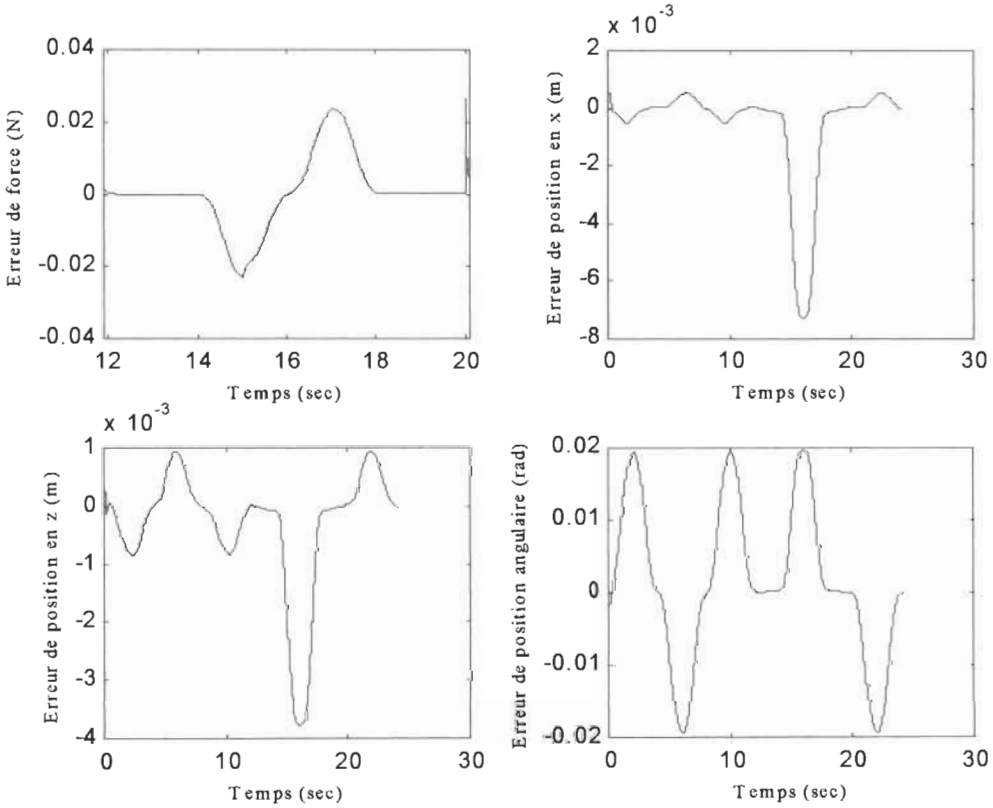

Figure G. 3 : Résultats obtenus suite à l'essai 6 (Rigidité de l'objet de 1kN/m et erreur cinématique de -5mm)

Résultats obtenus suite à l'essai 8 (Rigidité de l'objet de 1kN/m et erreur cinématique de 20mm)

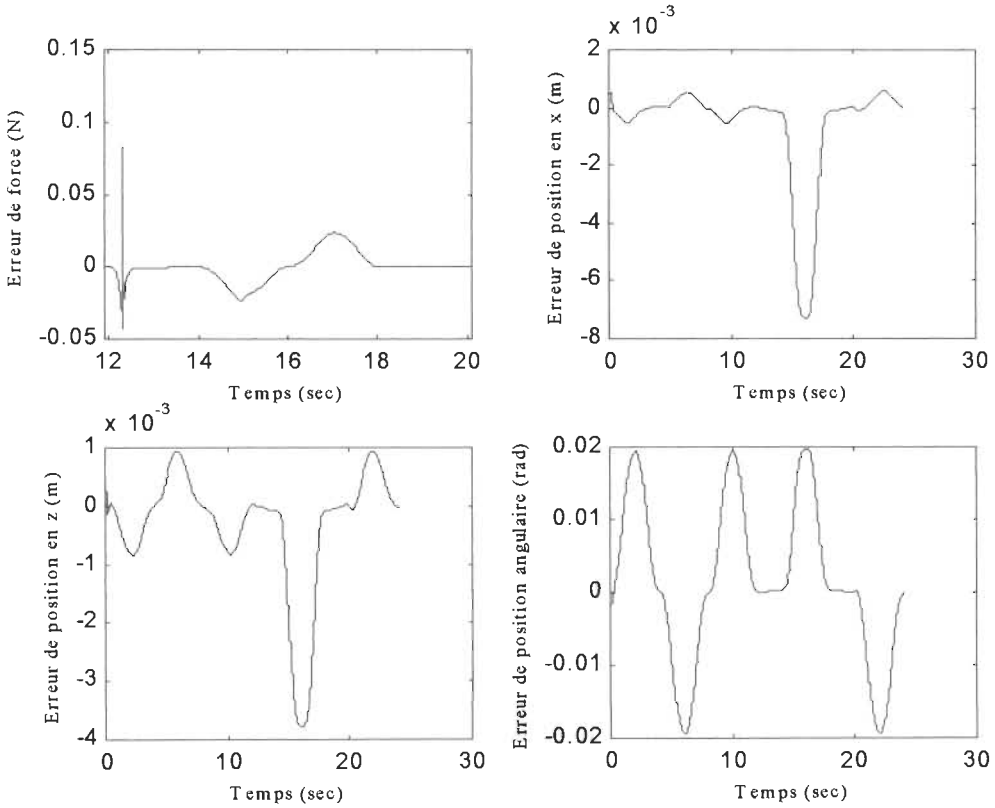

Figure G. 4 : Résultats obtenus suite à l'essai 8 (Rigidité de l'objet de 1kN/m et erreur cinématique de 20mm)

Résultats obtenus suite à l'essai 9 (Rigidité de l'objet de 2kN/m et erreur cinématique de -20mm)

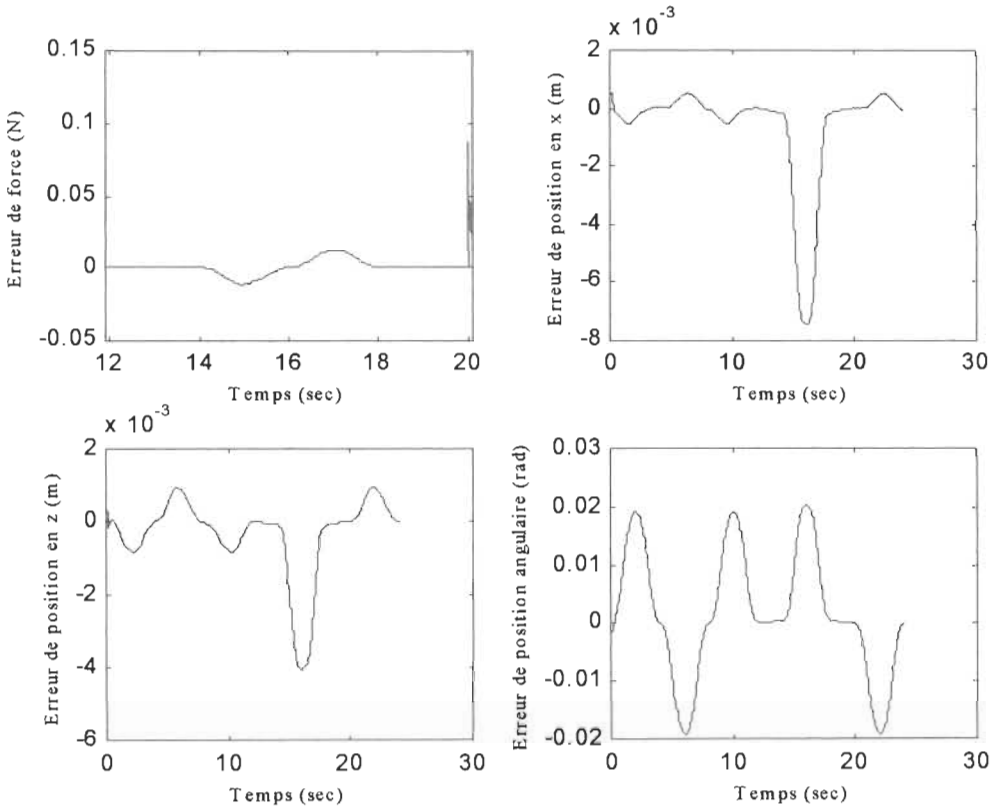

Figure G. 5 : Résultats obtenus suite à l'essai 9 (Rigidité de l'objet de 2kN/m et erreur cinématique de -20mm)

Résultats obtenus suite à l'essai 10 (Rigidité de l'objet de 2kN/m et erreur cinématique de -5mm)

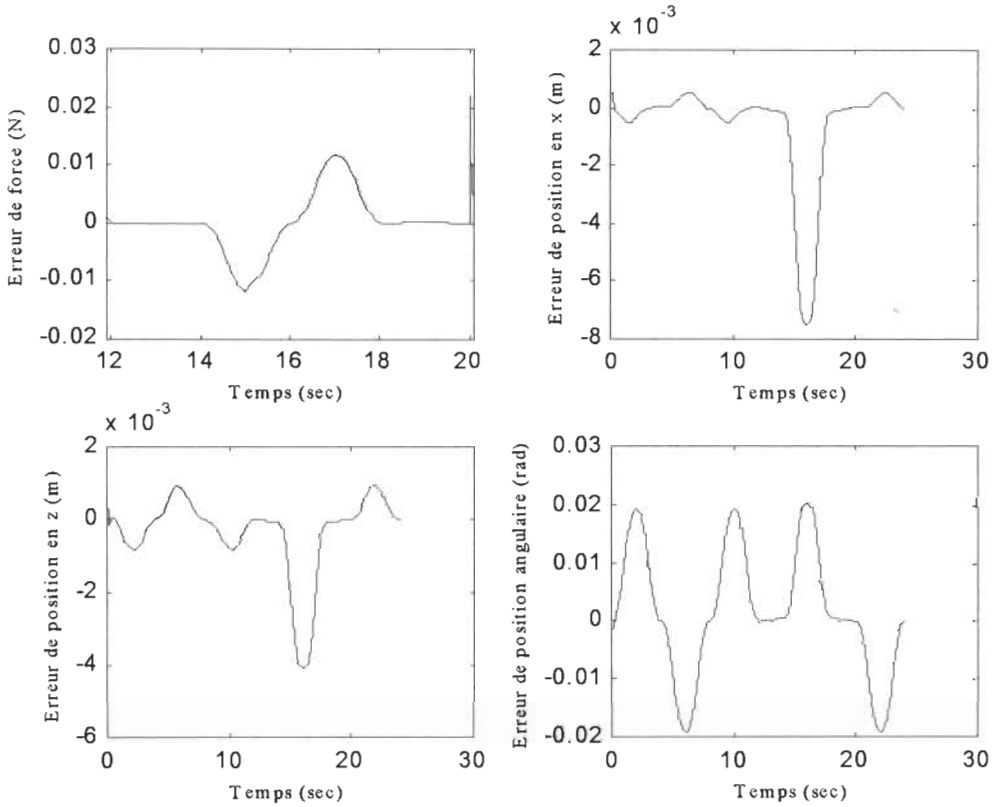

Figure G. 6 : Résultats obtenus suite à l'essai 10 (Rigidité de l'objet de 2kN/m et erreur cinématique de -5mm)

Résultats obtenus suite à l'essai 11 (Rigidité de l'objet de 2kN/m et erreur cinématique de 5mm)

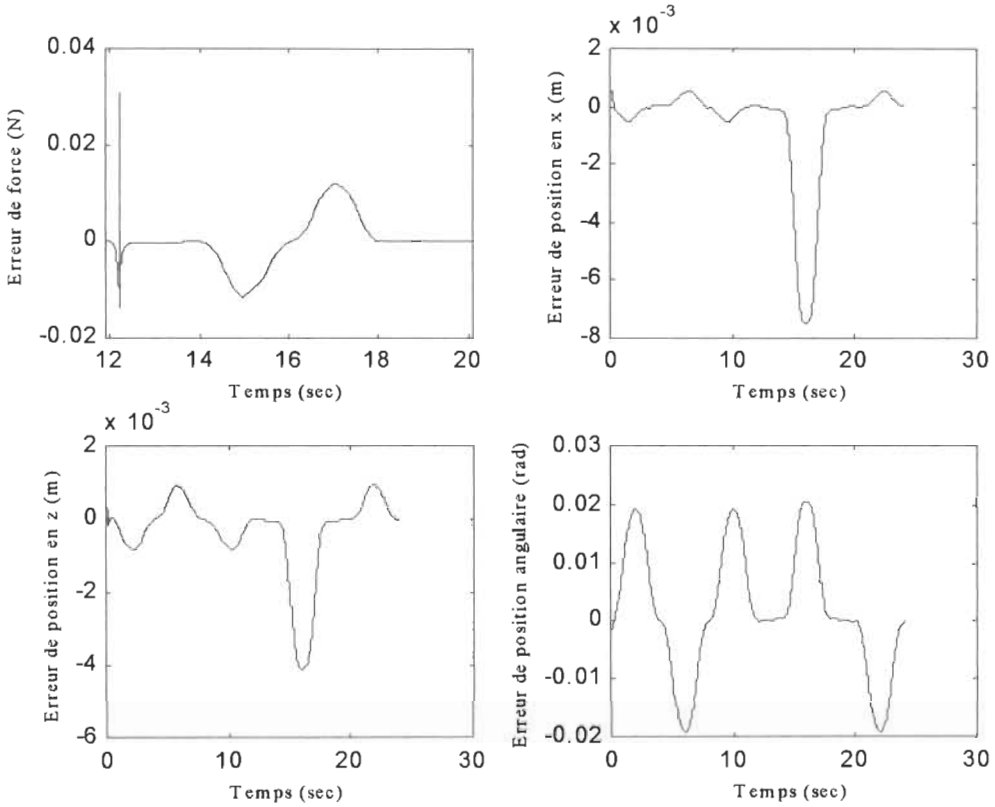

Figure G. 7 : Résultats obtenus suite à l'essai 11 (Rigidité de l'objet de 2kN/m et erreur cinématique de 5mm)

Résultats obtenus suite à l'essai 12 (Rigidité de l'objet de 2kN/m et erreur cinématique de 20mm)

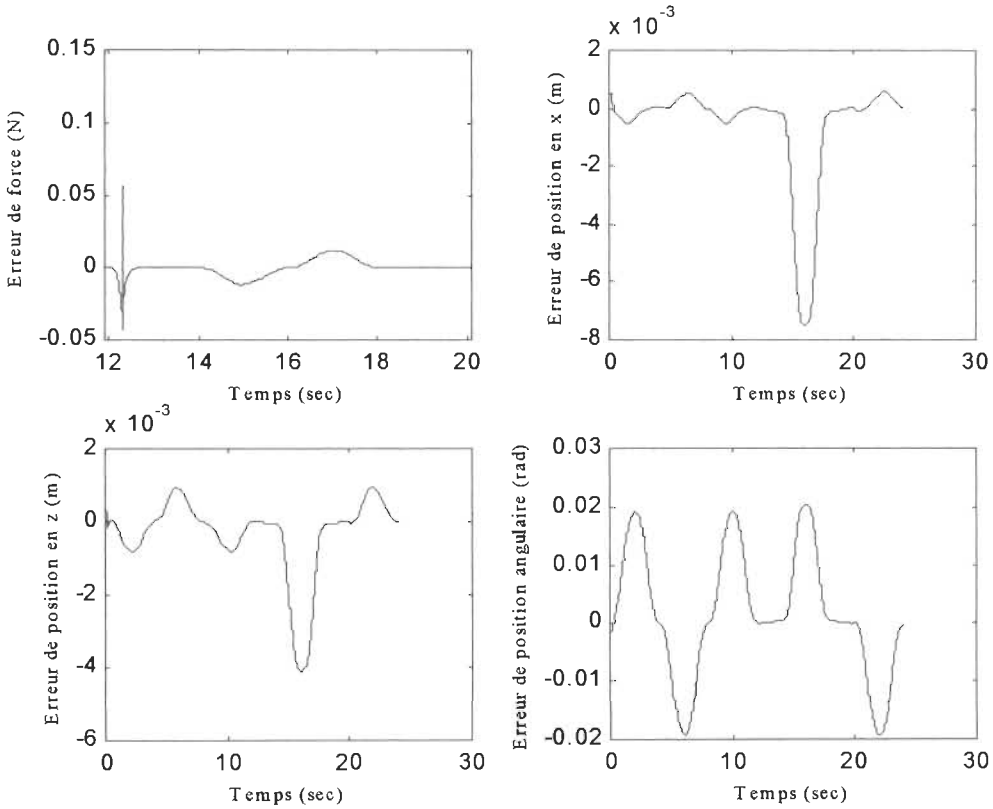

Figure G. 8 : Résultats obtenus suite à l'essai 12 (Rigidité de l'objet de 2kN/m et erreur cinématique de 20mm)